# Improving Online Kernel Machine Algorithms

by

**Jason P. Rhinelander**

A Thesis submitted to
the Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of
the requirements for the degree of
**Doctor of Philosophy**
in

Electrical and Computer Engineering
Ottawa-Carleton Institute for Electrical and Computer Engineering (OCIECE)
Department of Systems and Computer Engineering
Carleton University
May 2013

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

*Your file Votre référence*
*ISBN: 978-0-494-94544-5*

*Our file Notre référence*
*ISBN: 978-0-494-94544-5*

# Abstract

Online pattern recognition has two additional features when compared to offline pattern recognition. The first feature is training data arrives in a streaming fashion for a possibly infinite time. The second feature is that the process generating the training data can change over time.

The primary research problem addressed is the improvement of kernel machines operating in online environments. There are four linked subproblems that are interrelated and are addressed in this thesis. First, the computational efficiency of kernel machines is important in online applications. Second, the online adaptability of kernel machines is necessary when there is a change in the nature of the input data stream. Third, the estimation accuracy of kernel machines is important because of the online training methods used. Fourth, the limited memory of the online environment combined with *stochastic* gradient descent gives rise to truncation error in the kernel machine estimation.

A series of solutions are presented that address a subset of the proposed problems. For each proposed solution, experimental evaluation was conducted on simulated, benchmark, and/or real data and the advantages and disadvantages of each solution are discussed. As well, the contribution of the proposed solution is explained with references to existing literature.

Fully solving these four problems would allow online intelligent systems to have the same level of accuracy as offline batch systems but have a suitable computational complexity for online systems. This goal represents an asymptotic boundary as online processing will always have additional restrictions in terms of time and memory space when compared to offline processing. It is the finding of this thesis that there is significant room for improvement in the performance of online kernel machines and the methods in this thesis take some of the necessary steps towards the ideal boundary.

*For my parents, Elaine and Buzz, and my loving wife Cindy.*

# Acknowledgments

I would like to acknowledge the sources of help that I received while pursuing my Ph.D. in Electrical and Computer Engineering at Carleton University. I would like to thank my supervisor Dr. Peter X. Liu for his guidance and support during my time at Carleton. I would also like to thank all of my colleagues, past and present within our research lab, more specifically Ilia Polushin, Huanran Wang, Kun Wang, Xiaochun He, and Shichao Liu.

The research presented within the thesis is partially supported by the following sources: the Natural Sciences and Engineering Research Council of Canada, the Canada Research Chair program, and the Ontario Graduate Scholarship program.

I would like to thank all of my friends here in Ottawa and back in Newfoundland for all their support, specifically my friend Danny Lemay for ensuring that I had an adequate supply of coffee in the mornings. I would also like to thank my mother-in-law Catherine O'Driscoll and father-in-law Patrick O'Driscoll for their support during the good times and hard times. Finally, I would like to thank my wife Cindy O'Driscoll and for all her love and support and as always, her insight on research and life in general.

# Table of Contents

# List of Tables

# List of Figures

# List of Acronyms

| Acronyms | Definition |
|---|---|
| AS | Active Set |
| KM | Kernel Machine |
| KLMS | Kernel Least-Mean-Square |
| NORMA | Naive Online $R_{reg}$ Minimization Algorithm |
| PKM | Partitioned Kernel Machine |
| RBF | Radial Basis Function |
| RKHS | Reproducing Kernel Hilbert Space |
| SDC | Smooth Delta Corrected |
| S3 | Stochastic Subset Selection |
| SV | Support Vector |
| SVM | Support Vector Machine |

# List of Symbols

| Symbols | Definition |
|---|---|
| $\mathbf{x} \in \mathbb{R}^d$ | An input vector of dimensionality $d$ |
| $\mathbf{p}_t \in \mathbb{R}^t$ | A possibly infinite time-series of data values. |
| $m$ | The number of kernel expansion terms. |
| $\boldsymbol{\alpha} \in \mathbb{R}^m$ | Vector containing expansion term weights. |
| $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ | Kernel function. |
| $\|\bullet\|$ | Euclidean norm operator. |
| $r : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ | Similarity function. |
| $\phi : \mathbb{R}^d \to \mathbb{H}$ | Implicit casting of vector to RKHS. |
| $\gamma$ | Width parameter of the RBF kernel function. |
| $\eta$ | Learning rate for stocastic gradient descent. |
| $\lambda$ | Regularization term used in regularized risk functional. |
| $f : \mathbb{R}^d \to \{\pm 1\}$ | Output of kernel machine used for classification. |
| $f : \mathbb{R}^d \to \mathbb{R}$ | Output of kernel machine used for regression. |

# Chapter 1

# Introduction

Pattern analysis refers to the process of finding relations amongst a set of data. Pattern analysis finds extensive use in many modern intelligent systems that we find in everyday life. Examples of such intelligent systems are consumer electronics, industrial processing and manufacturing equipment, robotic control systems, software systems, and smart phones and tablets.

Intelligent systems will generally have one or more of the following characteristics: the ability to generalize from incomplete data, make decisions based on state and input, plan, reason, and learn from previous experience.

The inclusion of pattern analysis in an intelligent system enables the system to discover the relations within the data and then apply these relations to new data. If the relations exist as classification rules, regression functions, or cluster structures, the usual *statistical pattern recognition* approaches can be used [2].

## 1.1 Description of Kernel Machines

Kernel machines operate by the implicit mapping of an input pattern to a high-dimensional feature space, and then find an optimal separating hyperplane in that space [1, 3–5]. Support Vector Machines (SVMs) are but one member of the kernel

machine family. When SVMs are applied to classification or regression problems, they are referred to as support vector classification (SVC), and support vector regression (SVR) methods, respectively. SVMs are currently one of the best alternatives to other statistical learning methods found in general texts on artificial intelligence [6], and neural networks [7].

There are both advantages and disadvantages to using kernel machines in intelligent systems. Some advantages are:

- A SVM is trained to maximize the margin between different classes of data while simultaneously minimizing it's error. Maximizing the margin creates a robust classifier with good generalization characteristics even in the presence of sparse training data [1].
- A SVM is commonly formulated as a quadratic programming (QP) problem that is convex and therefore has a global optimum solution.
- SVMs can be formulated to be resistant to outliers in the training set.

Some disadvantages include:

- Training a SVM involves solving a quadratic programming problem where the number of variables to be optimized equals the number of training samples. Training with a large number of variables can be problematic in terms of memory requirements and training time.
- SVMs are kernel based approaches, and depending on the kernel function used, parameters may have to be defined a priori to training, through trial and error, or cross-validation.

## 1.2 Applications of Kernel Machines

SVMs are used for classification or regression, and have found use in systems for vehicle recognition [8], visual human motion recognition [9–12], road sign recognition [13], and motion blur detection [14]. SVMs are used in systems that have pattern recognition or regression requirements. For instance, Moustakidis et al. [15] applied SVMs for subject recognition from human gait signals. In the work of Zhu et al. [16], a novel SVM regression algorithm was proposed, and experiments were conducted with well known datasets from a robotic arm. Zhang et al. [17] investigated an algorithm that combined use of wavelet techniques when developing a kernel function for use with a SVM and the resulting algorithm was referred to as the "wavelet support vector machine".

SVMs have also been applied in the area of mechatronics. SVMs were recently used in the work of Liu et al. for the recognition of electromyographic signals [18]. Wang et al. used support vector machine regression to model friction in servo-motion systems [19]. In the research of Li et al, SVMs were used as part of a system for dynamics identification in haptic displays [20].

Real-time control systems can benefit from the use of SVMs. Yuan et al. use a SVM for the modeling and control of an electronic throttle valve [21]. In the work of Wang et al. an energy efficient SVM is used for learning and control of a bipedal walking robot [22]. Liu et al. make use of a SVM for the modeling and control of a permanent magnet synchronous motor [23]. There are many more examples of the use of SVMs in modeling and control, an interested reader can consult [24–30] for more examples.

In the work of Bellocchio et al. [31], the authors use Hierarchical Radial Basis Function (HRBF) networks for surface reconstruction. The HRBF neural network applies radial basis functions to regression, which is a similar approach to the use

of kernel machines using a RBF kernel function. In Ming et al. [32], the authors use kernel machines for human gait recognition. Further, in the research of Wan et al. [33], kernel machines were used for pattern recognition for a human brain-computer interface. These examples are just a small sampling of the various applications of SVMs. These applications are presented here because they are examples of problems that may require online and real time processing of information.

## 1.3 Illustrative Example

It is helpful to present an illustrative example of SVMs in the offline batch setting. For binary classification applications, the SVM will predict the class of an input vector. Due to the SVM formulation, the input data is classified by the SVM output either as $+1$ or $-1$.

The goal of a machine learning algorithm is to produce a hypothesis about the nature of observed data. In this example, the SVM will use ten discrete two dimensional data points to produce a decision hypothesis about the nature of the data and investigate the use of a SVM for identifying dangerous conditions in a robotic environment. Assume that there is an unmanned aerial vehicle (UAV) operating in a extreme environment. It is determined through simulation that the UAV will crash due to system failure or due to improper maneuvering under specific temperature and altitude conditions.

The process of developing the decision hypothesis is referred to as training the SVM. Chapter 2 will give specifics about the notation and terminology used throughout this thesis when discussing kernel machines such as SVMs. Below is the notation that is helpful with this particular example:

- $d \in \mathbb{N}^+$: The dimensionality of the input vector. In this example $d = 2$.

- $k \in \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$. The function $k$ is referred to as the kernel function for the SVM. In this example the radial basis function (RBF) kernel is used, $k(x_1, x_2) = e^{-\gamma \|x_1 - x_2\|^2}$.

- $\mathbf{x} \in \mathbb{R}^d$: A $d$-dimensional input vector. Input vectors can be used for training, for classification of new data, or both.

- $\alpha \in \mathbb{R}$: A scalar value or weighting that is used for producing the SVM output. The value of any particular $\alpha$ is determined from the training algorithm.

- $y \in \pm 1$: A label value that identifies which class the input vector belongs. Each data vector has an associated label. If the label is known, the input vector and label can be used to train or test the accuracy of the SVM. If the label is unknown, a trained SVM can be used to provide a prediction of the unknown label.

- $b \in \mathbb{R}$: A scalar bias value that adjusts the output of the SVM.

- $f_{SVM} \in \mathbb{R}^d \rightarrow \pm 1$: The output of the SVM. The output is a prediction of the class label for any particular input vector.

Readers wanting to become more familiar with the terminology surrounding SVMs and kernel machines are encouraged to read Chapter 2 or consult [1, 4]. A survey of SVM training methods can be found in [34] and interested readers are encouraged to review a survey researched by Sapankevych and Sankar [35] regarding regression applications using SVMs. Table 3 lists the simulation results. If the UAV crashed the simulation result is assigned a label of -1, otherwise it is assigned a value of +1. Table 3 lists the simulation trials in the order the experiments were performed.

The SVM is trained with the data presented in the above table and is placed into two dimensional vector/label pairs for training the SVM. The following format

**Table 3:** UAV Simulation Results.

| Simulation Trail Number | Temp (C) | Altitude (m) | Result |
|---|---|---|---|
| 1 | -12.0 | 80 | -1 |
| 2 | -5.5 | 110 | -1 |
| 3 | -5.0 | 60 | -1 |
| 4 | -2.0 | 80 | -1 |
| 5 | 0.0 | 100 | -1 |
| 6 | 1.0 | 90 | +1 |
| 7 | 2.5 | 105 | +1 |
| 8 | 5.0 | 85 | +1 |
| 9 | 10 | 140 | +1 |
| 10 | 12 | 110 | +1 |

is used: $\mathbf{x}_{trial}$ = [temperature, altitude], $y_{trial}$. For example, the third simulation trial would create the following vector/label pair, $\mathbf{x}_3, y_3 = [-5.0, 40], -1$. Equation (1) determines the output of the SVM. The summation term in Equation (1) is referred to as the kernel expansion of the SVM. The kernel expansion consists of $m$ terms. The size of $m$ corresponds to the number of training samples that become support vectors (non-zero $\alpha$ values). When training is performed, the values of each $\alpha_i$ are determined.

$$f_{SVM}(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^{m} \alpha_i k(\mathbf{x}, \mathbf{x}_i) + b\right) \tag{1}$$

The training of a SVM is accomplished by solving a constrained convex quadratic programming (QP) problem. Equation (2) gives the formulation of the QP problem. The greatest advantage of using the SVM over an artificial neural network (ANN) for classification is that the QP problem has a globally optimal solution [1]. After training, the SVM is a maximal margin classifier that will generalize as well as an ANN [1,3,4]. The parameter $C$ in Equation (2) allows for a trade-off between allowable

error and regularization in the SVM output.

$$\begin{aligned} \underset{\boldsymbol{\alpha}\in\mathbb{R}^m}{\text{maximize}} \quad & Z(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \tfrac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \\ \text{subject to:} \quad & 0 \leq \alpha_i \leq \tfrac{C}{m}, \forall i = 1 \ldots m \\ & \sum_{i=1}^{m} \alpha_i y_i = 0 \end{aligned} \tag{2}$$

Figure 1 gives the resulting output of the SVM (without computing the $sgn()$ function) when using different parameters for $\gamma$ and $C$. The parameters are usually found using minimum error on a separate test set of data or through the use of cross-validation. For illustration purposes in this example, the specific choice of $\gamma$ and $C$ is not as important as the computational complexity of finding suitable values for $\gamma$ and $C$ before the SVM is used to classify unknown data. Black 'x', pink circle, and green square markers represent the data points in Table 3 for which $\alpha_i = 0$, $\alpha_i \neq 0$ for $-1$ labels, and $\alpha_i \neq 0$ for $+1$ labels respectively.

Figure 2 illustrates a suitable parameterization for the SVM when predicting the outcome of the UAV flight. The choice of $\gamma$ and $C$ give accurate results while yielding a smooth decision surface for good generalization between classes.

The resulting SVM can be implemented in an embedded real-time operating environment on the UAV in this example. The SVM bias, $b$, and all $\mathbf{x}_i$, and $\alpha_i$ corresponding to $\alpha_i \neq 0$ need to be stored in computer memory. Any $\mathbf{x}_i$, corresponding to $\alpha_i \neq 0$ are referred to as the support vectors (SVs) of the SVM. Figure 3 shows the structural organization of the SVM used in the embedded environment of UAV. The structure of the SVM is similar to a single layer ANN, but the most significant difference is the training algorithm used.

All the problems addressed in this thesis have a common theme. The kernel machine is operating in an online environment. It must be efficient in its execution

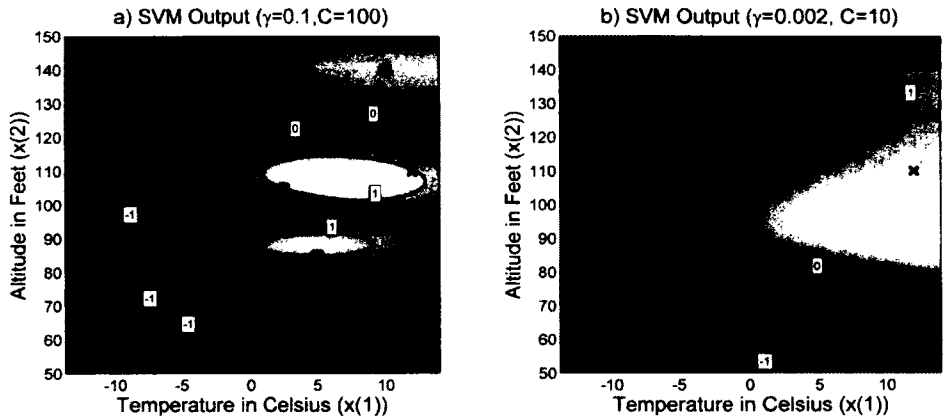


**Figure 1:** a) The RBF kernel function parameter $\gamma$ is too large resulting in too low of a similarity between stored vectors. In this case while the decision surface accurately represents the training data, this SVM will not generalize well to unknown data. The decision surface is too complex and requires more regularization. b) The regularization parameter, $C$, is too small yielding a SVM with a smooth decision surface, but the SVM does have an error on the training set. The error on the training set indicates that the SVM does not provide a good estimate for the UAV simulation data.

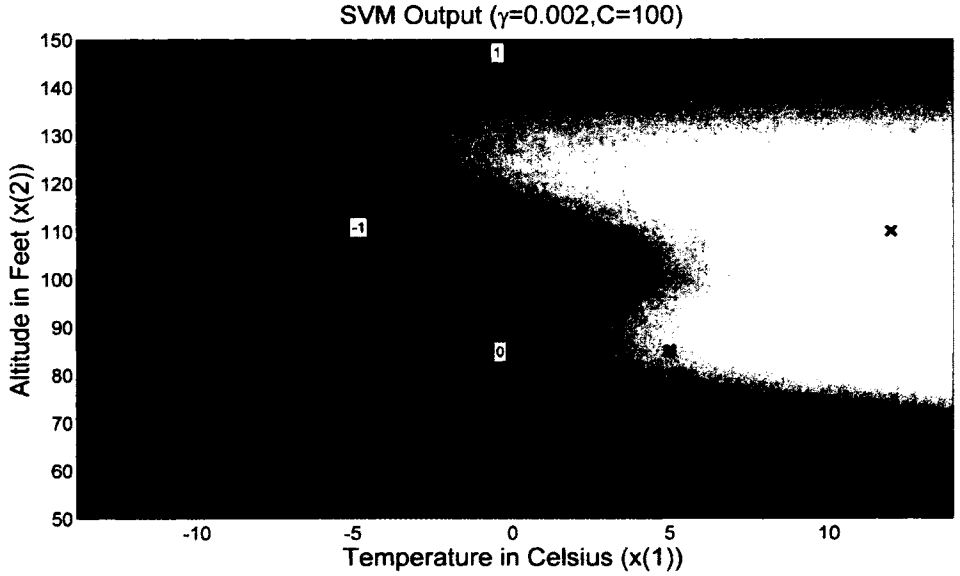


**Figure 2:** SVM output across all temperature and altitude.

but also must adapt and learn from new data arriving in a sequential fashion. The computational complexity in evaluating Equation (1) scales with the number of SVs

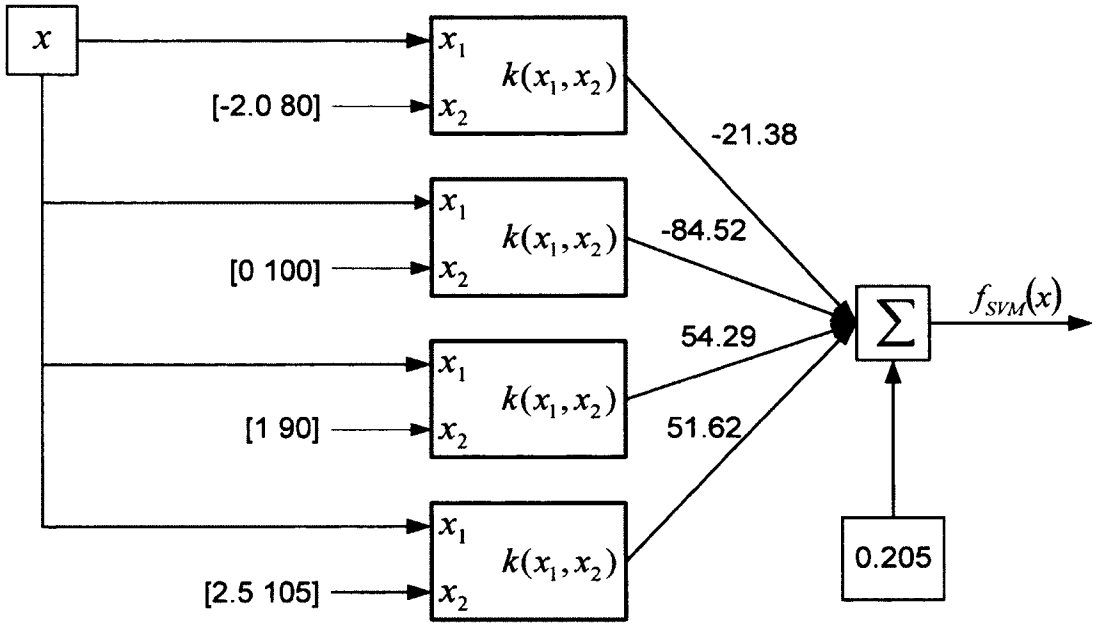


**Figure 3:** Structural organization of the SVM.

stored in the kernel machine ($O(m)$). It is evident from Equation (2) that in order to solve the quadratic programming problem, all data must be present in the form of kernel function evaluations between data vectors. Additionally the tuning of $\gamma$ and $C$ may be needed in order to preserve the optimality of the SVM, which is computationally inefficient when using additional test sets or n-fold cross-validation techniques.

### 1.3.1 Special Considerations of Online Environments

Online learning is necessary in situations when the data arrives in an infinite stream. In this situation, there is no guarantee that the properties of the arriving data will remain fixed or in other words, the arriving data may have non-stationary characteristics. The online learning scenario means the kernel machine can adapt to data that can change over time (i.e. non-stationary data). Figure 4 illustrates online learning with kernel machines, and the figure is the most computationally simple learning

scenario. If at each time step multiple $\alpha$ values were updated, more computational resources would be required.

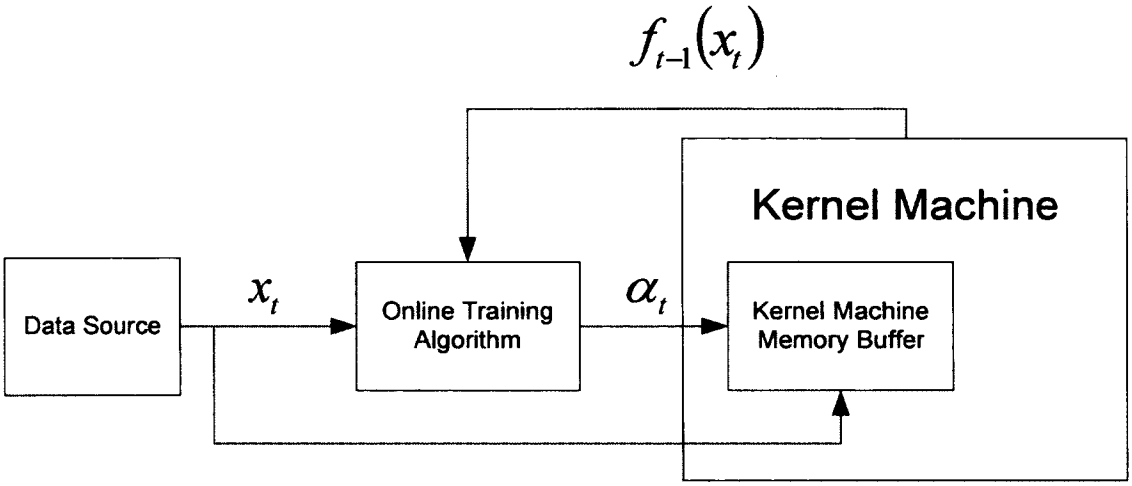


**Figure 4:** General view of kernel machines in online environments. At time $t$, new data arrives and the training algorithm will produce a scalar weight value, $\alpha_t$ for the support vector $x_t$. The training algorithm will also use the kernel machine output for determination of $\alpha_t$.

The kernel function used by a kernel machine allows the input data to be implicitly mapped to a high dimensional space. The type of kernel function used and the kernel parameterization needs to be determined. The parameterization of the kernel function depends on the nature of the input data.

The traditional approach for training support vector machines has two main flaws when used in an online setting. The first flaw is that the traditional training approach considers all data at once, with no concept of "when" the data set was acquired. It is important for any online learning algorithm to be able to adapt to changes over time [36]. The second flaw is that the traditional training approach is usually computationally expensive. It is important that the kernel machine is computationally efficient yet accurate enough to be used in the online system.

## 1.4 Literature Review of Online Kernel Machines

The most simple and effective method for using a kernel machine in an online environment is to make use of a sliding window of size $m$, on the data. Sebald and Bucklew [37] introduce decision feedback using a windowed SVM for non-linear equalization. In their work they discuss the benefits of an online support vector machine algorithm verses a quadratical program training method.

In the work of Cauwenberghs and Poggio [38] an incremental-decremental algorithm for training support vector machine is presented. Their algorithm introduces the idea of adiabatic increments for the support vector weights. While the approach is accurate, there is no guarantee for the number of updates required. A projection method was presented by Castó and Opper [39] in terms of Gaussian processes (similar to SVMs). The drawback to Castó and Opper's algorithm is the need for a matrix multiplication in each step and the dimensionality of the matrix scales with the number of support vectors used [40]. Other examples of incremental algorithms can be found in [41] and [42].

The perceptron algorithm was developed in 1969 by Rosenblatt [43] and directly computes a separating hyperplane between two sets of linearly separable data. The largest benefit of batch SVMs is that they maximize their margin of separation between classes of data. The large margin in the input space implies sparsity in the number of terms in the kernel expansion [44]. Some other algorithms that use perceptron-like approaches that enforce a large margin or use kernel functions are, the budget perceptron [45], the forgetron [46], the kernel adatron [47], the work of Cesa-Bianchi and Gentile [48], the kernel perceptron [2], the Relaxed Online Maximum Margin Algorithm (ROMMA) [49], and the Approximate Large Margin Algorithm (ALMA) [50].

Kivinen et al. [40] introduce the Naive Online $R_{reg}$ Minimization Algorithm

(NORMA) as an efficient method for classification, regression and novelty detection [40]. The NORMA uses linear loss and a time decay regularization technique to enforce simplicity in the kernel machine's output. The NORMA training algorithm has a computational complexity of $O(m)$, making it useful for online applications. The use of time decay regularization produces a biased estimator and can negatively impact performance of the kernel machine [51].

Implementation of kernel machines can also follow a parallel to adaptive filter theory. Engel et al. developed the Kernel Recursive Least-Squares Algorithm (KRLS) [52] which has an explicit sparsification method for providing regularization of the kernel machine. The computational complexity of training KRLS is $O(m^2)$. The kernel least-mean-square algorithm by Liu et al. [51, 53] is similar to the NORMA in that it trains the kernel machine using stochastic gradient descent, but requires no explicit regularization. The time complexity for training KLMS is stated as $O(m^2)$ in [53] but stated as $O(m)$ in [51]. KLMS with a training time complexity of $O(m)$ will be considered in this thesis. Due to their computational complexity the NORMA and KLMS will be used as starting points when addressing the problems described in Section 1.5.

## 1.5 Problems Addressed

Section 1.3 describes an example of a SVM used as a solution to a classification problem. Without the loss of generality, all kernel machines must evaluate a kernel expansion to produce an output. Equation 3 describes a generic kernel expansion equation.

$$f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i k(\mathbf{x}, \mathbf{x}_i) \tag{3}$$

The primary goal of this research is to improve upon the performance of kernel

machines in online environments. Equation (3) shows that the computational efficiency and the memory efficiency of a kernel machine is bounded by the number of expansion terms that make up the kernel machine. In the case of online environments, the computational complexity and memory complexity will grow over time.

Some fundamental concepts must be introduced before formalizing the problem addressed by this thesis. The goal of pattern recognition is to develop a hypothesis about the nature of observed patterns. In other words, given a pattern and label pair, $\mathbf{x}, y$ ($\mathbf{x}, y$ can be thought of as a single pattern as well), the hypothesis is produced by the function $f$. The generalized loss function is described by the real-valued function $l(\mathbf{x}, y, f(\mathbf{x}))$. The loss function is synonymous to an error function and can take many forms but it must be a non-negative function.

To avoid loss of generality is assumed that the pattern and label are defined on the sets $X$ and $Y$ respectively and there exists a probability distribution $P(\mathbf{x}, y)$. The risk of the hypothesis function $f$ is defined in Equation (4). The risk cannot be calculated because $P(\mathbf{x}, y)$ is unknown. The *empirical* risk of a given function can be calculated by the use of Equation (5) which references a collected test set of $k$ pattern and label pairs.

$$R[f] = \int_{X \times Y} l(\mathbf{x}, y, f(\mathbf{x})) dP(\mathbf{x}, y) \tag{4}$$

$$R_{emp}[f] = \frac{1}{k} \sum_{j=1}^{k} l(\mathbf{x}_j, y_j, f(\mathbf{x}_j)) \tag{5}$$

Equation (5) gives a method for measuring the accuracy of the hypothesis function on a specific test set of data. It is important to note that Equation (5) does not give insight if the function $f$ is *over fitting* the given data. Section 2.3 gives more details on the concepts of risk and loss. Regularized risk is introduced so that both the loss and *complexity* of the function $f$ is considered. The regularized risk is described by

Equation (6).

$$R_{reg}[f] = R_{emp}[f] + \frac{\lambda}{2}\|f\|_H^2 \tag{6}$$

The term $\|f\|_H^2$ is a measure of the function complexity in a possibly infinite dimensional Reproducing Kernel Hilbert Space (RKHS). The parameter $\lambda$ allows the amount of complexity to be adjusted. Kernel functions represent dot-product in RKHS and allows for the casting of classification and regression tasks from input space to RKHS. The importance of Equation (6) becomes obvious that minimizing the empirical risk reduces the loss and also the complexity of the hypothesis. Equation (6) is convex since the loss function is non-negative and $\|f\|_H^2$ is also convex.

The Representer theorem as introduced in the work of Kimeldorf and Wahba [54] and generalized in the work of Schölkopf and Smola [1] makes an important statement about the nature of minimizing the regularized risk functional. The Representer Theorem states that if there is a function $f \in H$ such that the regularized risk is minimized, then $f$ can be expressed as in Equation (7).

$$f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i k(\mathbf{x}, \mathbf{x}_i) \tag{7}$$

Equation (7) is exactly the same as the kernel expansion described in Equation (3). The Representer theorem does not indicate how many of the $\alpha$ terms will be zero, but it does explicitly state that the representation will span the set of training vectors. It is therefore implied that in the online setting illustrated in Figure 4, that as time progresses, the number of expansion terms, $m$, will also grow without bound which is computationally infeasible for online applications. The Representer theorem provides a lower bound of $O(m)$ on the computational complexity for training and evaluating kernel machines.

In addition to Equation (6) and the Representer theorem, the nature of online

learning offers significant challenges for the online use of kernel machines. Time is now a factor when considering the input patterns used for training and testing, and also the functional dependency between pattern labels and input patterns. For example in terms of probabilities, $P(\mathbf{x}) \neq P(\mathbf{x}, t)$ and $P(y|x) \neq P(y|x, t)$ indicate concept drift. The singular problem of online learning with kernel machines can be broken down into four inter-related subproblems. This thesis presents novel work that focuses on the four problems described in Sections 1.5.1 to 1.5.4.

### 1.5.1 Computational Efficiency

The example in Section 1.3 started with ten training vectors. After training, only 4 vectors were used to determine the SVM output for any input vector. It is very common for support vector machines to have a smaller subset of support vectors compared to the total number of training vectors. Throughout this thesis I will often refer to the number of support vectors by the variable $m$. Assuming the computational complexity of evaluating the kernel function is $O(1)$ then the computational complexity of evaluating Equation (1) scales in $O(m)$. Linear time complexity can be an issue in online systems if the number of support vectors grow linearly with time. The Representer Theorem as introduced in the work of Kimeldorf and Wahba [54], implies that the number of support vectors can grow linearly with the number of training vectors in the training set [40]. Online environments deal with an infinite stream of data and therefore the time required for kernel machine evaluation in such an environment can also grow without bound.

The process of training a kernel machine involves finding the solution to a QP problem in a batch fashion. General purpose QP solvers can be used for finding the QP solution to Equation (2), but are more computationally expensive than specialty algorithms. Classical QP solvers operate with a computational complexity of $O(n^2)$ to $O(n^3)$ [55]. Sequential minimal optimization (SMO) introduced by Platt [56] is

the most popular QP solver for off-line batch SVMs. SMO has a computational complexity between $O(n)$ and $O(n^2)$ [56].

Stochastic gradient descent on a loss functional is often used when training online kernel machines. Two important kernel machine training algorithms that use stochastic gradient descent are the Naive Online $R_{reg}$ Minimization Algorithm (NORMA) [40], and the Kernel Least-Mean-Square (KLMS) algorithm [51, 53]. As with SVMs, the NORMA and KLMS trained kernel machines have $O(m)$ complexity for their evaluation. Training a kernel machine with the NORMA or KLMS is accomplished with $O(m)$ computational complexity. The NORMA and KLMS scale in terms of the number of support vectors that are retained, and as the case with batch SVMs, the number of support vectors can grow linearly with the number of training vectors used.

In the online learning scenario there is an infinite stream of data. The training and evaluation computational complexity can therefore grow without bound, posing a problem for online systems where computational resources can be limited.

### 1.5.2 Adaptability

When using kernel machines in offline environments there are many useful techniques for tuning various parameters of the kernel machine. Cross-validation, leave-one-out error estimates, and the use of error rates on a separate set of data are a few such approaches [1,51]. In online systems, these types of system optimizations are far from practical, if not impossible, without failing real-time requirements.

Another problematic area for online use of kernel machines is *concept drift* [40,57]. The main idea behind kernel machines is that they produce a hypothesis about the input based on past inputs. Concept drift is a change in the underlying process that produces the input for the kernel machine. There are two types of concept drift. First, the nature of the data can change slowly over time and in this case the data

is referred to as *drifting*. Second, the nature of the data can change abruptly at a specific time and in this case the data is referred to as *switching*. Concept drift poses a significant problem for effective use of kernel machines in online situations because SVMs have no capability to adapt to concept drift. When training SVMs, all training data is treated equally valid when solving the QP problem, and in this case it makes little sense to use a batch learning algorithm [40].

Adapting kernel parameters (depending on the choice of kernel used) will have a direct effect on the regularized risk, $R_{reg}[f]$ of the kernel machine as it can affect both the empirical risk, $\frac{1}{k}\sum_{j=1}^{k} l\left(\mathbf{x}_j, y_j, f\left(\mathbf{x}_j\right)\right)$ and the complexity of the kernel machine in RKHS, $\frac{\lambda}{2}\|f\|_H^2$. The type of kernel function used with a kernel machine will depend on both the error incurred and the nature of the data.

### 1.5.3 Accuracy

Efficient online kernel machine training algorithms such as the NORMA and KLMS have to balance between approximation accuracy and adaptability. Both the NORMA and KLMS have a learning rate parameter [40, 53] that can be set depending on the required performance of a given situation. Determining the learning rate in the online learning scenario can be done offline with cross-validation.

The NORMA and KLMS are truly online algorithms for the training of kernel machines because they attempt to minimize the instantaneous risk of the kernel machine, rather than the empirical risk, $R_{emp}[f]$. The instantaneous risk as defined in the NORMA [40] by Equation (8).

$$R_{inst}[f_t, x_t, y_t] = l\left(x_t, y_t, f\left(x_t\right)\right) + \frac{\lambda}{2}\|f\|_H^2 \tag{8}$$

Equation (9) gives the general idea behind stochastic gradient descent that is used to train the NORMA [40]. It will be shown later in Section 4.2 that KLMS also

uses the same type of stochastic gradient descent. The update of $f_{t+1}$ at time $t$ only depends on the current input pattern, label and kernel expansion.

$$f_{t+1} \leftarrow f_t - \eta_t \frac{\partial \left( R_{inst} \left[ f \right] \right)}{\partial f} \tag{9}$$

The learning rate, $\eta_t$ can also be adapted using cross-validation and can provide excellent results but is computationally expensive. There is limited literature in terms of true online learning rate adaptation for kernel machine algorithms. In the work of Kivinen et al [40] it is suggested that learning rate decay be employed for better online results. While Vishwanathan et al. [55] introduce a learning rate adaptation algorithm referred to as Stochastic Meta-Descent (SMD) that allows the online adaptation of the learning rate for the NORMA. The use of SMD requires more computational effort but the NORMA using SMD still has a computational complexity of $O\left(m\right)$. As discussed in the literature [1, 40, 51, 53, 55], adaptation of learning rate can improve the accuracy of the online kernel machine.

Both the NORMA and KLMS are based upon stochastic gradient descent and improving upon this basic building block is a significant open ended research problem. Equation (9) shows that there is an opportunity for improvement of the minimization of the regularized risk of a kernel machine trained using the NORMA of KLMS. Gradient descent methods rarely converge to an optimal solution in one step as shown in Equation (9). When this problem is addressed it can improve upon online kernel machine accuracy and still keep the computational complexity within $O\left(m\right)$.

### 1.5.4 Memory Limitations

Similar to the computational complexity of online kernel machines, the memory space complexity is also $O\left(m\right)$. Due to the Representer Theorem [54], the amount of required computer memory required for storing the support vectors of a kernel machine

will grow without bound [1, 40]. In the case of limited memory space one or more support vectors must be discarded to make room for a new support vector. There exists a trade off that becomes evident when comparing Equations (1) and (2) in Section 1.3. In Equation (1), it can be seen that the accuracy of the kernel machine is dependent on the number of support vectors stored. In the batch example from Section 1.3, solving the QP problem described by Equation (2) can be accomplished by ignoring any $\alpha$ that will be removed to make space in computer memory.

In the online case, all online training algorithms for kernel machines use the kernel machine evaluation to determine the new value of $\alpha$. In the work of Kivinen et al., the authors define the concept of *truncation error* for kernel machines with finite memory. Truncation error poses a significant problem for online kernel machines due to the fact that existing training methods do not account for finite memory during the learning process.

Given the kernel expansion in Equation (3), it can be seen that the memory complexity also grows with $O(m)$. From a practical engineering perspective, the use of a finite sized memory buffer requires the replacement of an existing expansion term. The NORMA and the KLMS algorithm do not address the problem of limited memory in their formulation. The problem of truncation error is formalized with the difference between two risk functionals in Equation (10).

$$\Delta = \|R[f_{t+1}] - R[f_t]\| \tag{10}$$

At time $t$ and $t+1$ there will be a total of $m$ terms in the kernel expansion and $\Delta > 0$ represents the difference in risk minimization between $f_{t+1}$ and $f_t$. If the risk could be calculated directly and efficiently the problem of truncation error would be solved. Empirical risk could be substituted instead of the risk functional, but this is not computationally feasible. The loss for each vector stored in the buffer would

have to be calculated resulting in $O(m^2)$ evaluations. Accuracy verses computational complexity must be considered when addressing the problem of truncation error.

### 1.5.5 Connection Between Problems Addressed

There are connections between the four problems discussed in Sections 1.5.1 to 1.5.4. Figure 5 illustrates the relationship between the four problems on how they relate to online environments and the characteristics of kernel machines. The blue shaded bubbles in Figure 5 are the problems related to kernel machine evaluation and implementation and are heavily dependent on the number of terms in the kernel expansion. The green shaded bubbles relate to problems related to effective training of the kernel machine and are therefore directly linked to risk minimization. The Representer theorem is directly linked to both the implementation and training of online kernel machines. The Representer theorem is linked to the implementation of kernel machines since it implies that the number of expansion terms can grow linearly with time [40]. It is also linked to risk minimization because it is the training algorithm which minimizes the regularized risk of the kernel machine.

The characteristics of the online environment therefore have a direct effect on all four problems due to the Representer theorem. The online environment can provide an infinite stream of data which affects both the computational efficiency and the memory complexity of the kernel machine. The online environment can also exhibit concept drift which has an effect on the minimization of the regularized risk.

The linkages in Figure 5 only provide a high level view of how the problems addressed fit into a online risk minimization framework. There are also dependencies between the problems directly. For example, the use of kernel parameter adaptation or learning rate adaptation can affect the accuracy and the size of the kernel expansion. As well, virtually all online kernel machine training algorithms must evaluate the kernel expansion, therefore the efficiency of the kernel machine evaluation has a direct

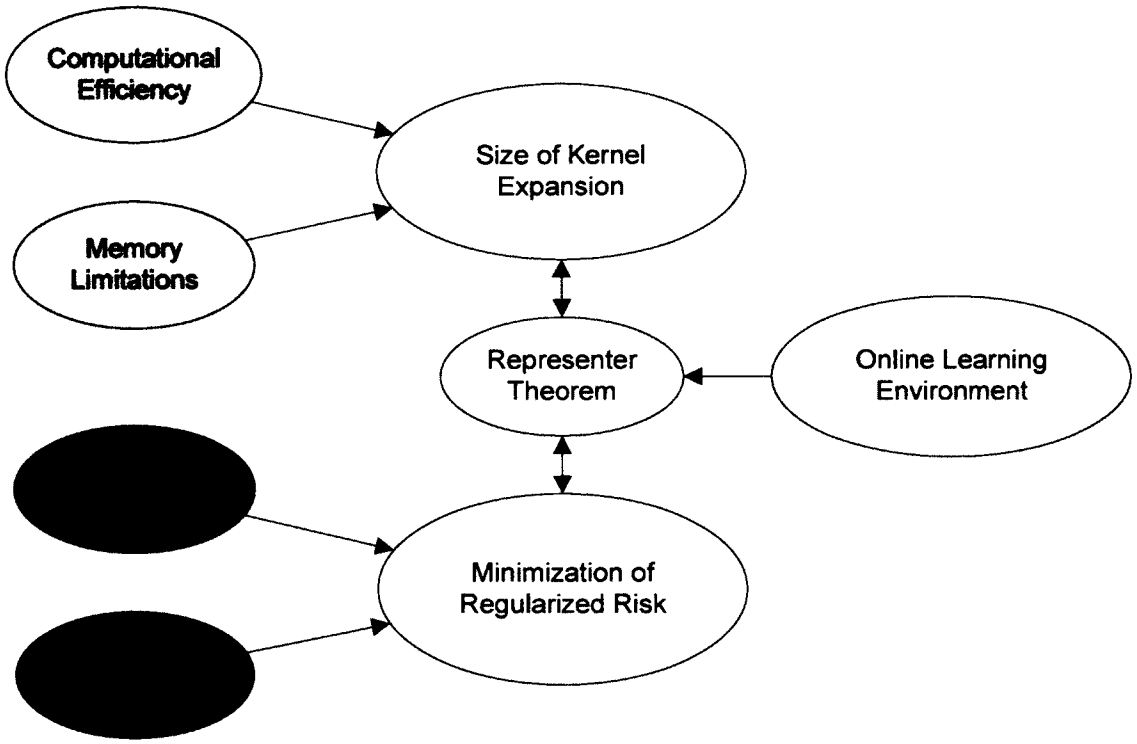


**Figure 5:** A high level view of how the online learning environment relates to the problems addressed in this thesis.

effect on the efficiency of the training of the kernel machine.

## 1.6 Thesis Scope

The general goal of this thesis is to develop computationally efficient, accurate, and adaptable pattern recognition algorithms for intelligent systems in online environments. More specifically, this thesis seeks to improve upon existing online training algorithms for kernel machines by addressing the problems described in Section 1.5. Kernel machines have proven to be among the best pattern recognition algorithms in an offline environment and online use of kernel machines is an open research question. The following describes the methodology that was used in the research conducted in this thesis.

After a review of existing literature and analysis of the problems outlined in Section 1.5, it became evident that there was a need for a novel contribution to this subject area. The algorithms described within this thesis are tested against a baseline algorithm for comparison of performance. Comparisons are performed against simulated data sets and benchmark data sets that are commonly used in the field of pattern recognition.

When describing the computational complexity of an algorithm, execution time is tested and asymptotic bounds are determined. When examining the performance of kernel machine classifiers, percentage of misclassification or correct classifications are given. For kernel machine regression, the use of total error against the test data is compared. In the case of novelty detection, contour plots are provided for both the baseline algorithm and the improved kernel machine. The following is an outline of the rest of this thesis including an explanation of the contribution, advantages, and possible disadvantages of the algorithms proposed.

Chapter 2 provides background, preliminary theory and notation that is useful when discussing kernel machines. This chapter also describes two key algorithms (KLMS and the NORMA) for the use of kernel machines in online environments. If the reader is familiar with the topic of kernel machines, these sections can serve as a reference. The motivation for online kernel machine algorithms is described with a survey of the innovations in the field.

Chapter 3 introduces the stochastic subset selection (S3) algorithm. The S3 algorithm has been published in [58]. The S3 algorithm addresses the problem of efficiency of evaluation and training described in Section 1.5.1. The use of a stochastic subset of support vectors for expansion evaluation allows for a kernel machine to be efficient in its execution and training while still being adaptable to changes in the input. The size of the stochastic subset is chosen a priori depending on the computational complexity required. The disadvantage to using a stochastic subset for kernel machine

evaluation is that noise is introduced in the output of the kernel machine. The S3 algorithm was validated through experiments involving simulated data, a benchmark handwritten digit recognition [59] dataset, and a benchmark online non-stationary dataset [60]. Computational complexity of the S3 algorithm is also validated using simulated data.

Chapter 4 describes the partitioned kernel machine (PKM) algorithm [61]. The PKM algorithm improves on the problem of accuracy as discussed in Section 1.5.3. The use of a similarity measure for training the kernel machine allows for adaptation to the nature of the input. While the use of a similarity measure increases the computational requirement of training the kernel machine, it scales to the same computational complexity of the existing online kernel machine algorithms. In addition, validation of the PKM algorithm is performed on a simulated time series function and a benchmark chaotic time series used commonly in adaptive filter literature [51].

Chapter 5 describes the Smooth-Delta-Corrected-KLMS (SDC-KLMS) algorithm. The SDC-KLMS addresses the problem of limited memory situations as discussed in Section 1.5.4. The SDC-KLMS algorithm improves upon the traditional KLMS algorithm [53] by adapting the training procedure to compensate for buffer truncation error and also adding in-place updates to the traditional KLMS algorithm. Stability of the KLMS training algorithm is considered when truncation error is present. The issue of truncation error was raised in the work of Kivinen et al. but was not addressed in the algorithm design of the NORMA. For the KLMS algorithm in the work of Liu et al. [53] problems associated with finite memory were not addressed. Validation of the SDC-KLMS algorithm is accomplished with a comparison of error rates with the standard KLMS algorithm. The first dataset to be tested was a simulated time series. The second dataset was the same benchmark chaotic time series from chapter 4. The third dataset to be tested was real-world 3D trajectory data collected from a haptic device.

Chapter 6 describes an online algorithm for the adaptation of the RBF kernel machine parameter. The RBF kernel is a popular choice for the implementation of non-linear kernel machines. Input space statistics are used for input to a fuzzy controller that adapts the RBF kernel parameter. Kernel parameter adaptation is a method for adapting the online kernel machine to changes to the nature of the input. Kernel parameter adaptation addresses the problem of adaptability described in 1.5.2. Experiments were conducted using the fuzzy logic controller with a kernel machine for data reconstruction over a lossy communications channel. Dropped data samples were simulated using a random process. Two different signals were used for testing, a simulated sinusoidal time series and the same haptic trajectory as used in Chapter 5. The fuzzy logic control provided a better error rate when reconstructing dropped data in the case of the sinusoidal time series.

Chapter 7 presents the Active Set (AS) algorithm for the training of online kernel machines that can be used for anomaly detection, density estimation and mode tracking. The AS algorithm deviates from the other chapters in this thesis because it optimizes a quadratic programming problem that was first proposed by Scholköpf et al. [62]. In [62] a special version of SMO is used to train what the authors refer to as a $\nu$-single class SVM. In a similar fashion to SMO, the AS algorithm performs optimization of the QP problem, but does so with online processing in mind. The training of a single class $\nu$-SVM using the AS algorithm is faster than using Matlab's built in medium-scale QP solver (quadprog()). The AS algorithm was used to estimate the density of a 2D simulated dataset.

Chapter 8 is a summary of the findings from each chapter and an overall conclusion on the state of online learning with kernel machines.

The academic contributions of this thesis are listed below:

1. A computational efficient subset selection algorithm for online kernel machines used for classification.

2. An adaptable and accurate partitioning algorithm for online kernel machines used for function approximation.

3. An algorithm for error correction for online kernel machines operating in finite memory environments.

4. An algorithm for adaptable kernel parameter tuning.

5. An online algorithm for training kernel machines for novelty detection and mode switching.

# Chapter 2

# Background, Preliminary Theory and Notation

## 2.1 Mathematical Notation

A consistent mathematical notation is used throughout this thesis. Scalar variables are represented by *italicized* lowercase letters, and vector variables by **bold** lowercase letters while matrices are *italicized* capitalized letters. All vectors are assumed to be column vectors. When referring to a component dimension of an input vector, round bracketing is used. For example, the $i$th dimension of the vector $\mathbf{x}$, is stated as $\mathbf{x}(i)$. Subscripting is used when discussing the time sequence of data, describing the source of the data, or when referring to a specific item within a data structure, such as a buffer.

The following is a description of some of the functions and variables used in this paper:

- $\mathbf{x} \in \mathbb{R}^d$: A vector of dimensionality $d$. $\mathbf{x}$ can be an unknown vector to serve as the input for the kernel machine, or it can be a vector to be used for training the kernel machine.

- $\mathbf{p}_t \in \mathbb{R}^t$: A possibly infinite time-series of data values.
- $m$: The number of support vectors or kernel basis functions stored within the kernel machine buffer.
- $n$: The size of the training set for an offline SVM.
- $y \in \mathbb{R}$ or $\pm 1$: The desired value or binary label that the kernel machine is trained with for kernel machine regression or classification respectively.
- $\boldsymbol{\alpha} \in \mathbb{R}^m$: A vector of weights that scale the kernel functions centered on the observed input vectors. Sometimes for convenience, when referring to the $i$th component of $\boldsymbol{\alpha}$, it is written $\alpha_i$.
- $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$: Defines a kernel function that represents a dot product in Reproducing Kernel Hilbert Space (RKHS), between the two input vectors, $\mathbf{x}_1$ and $\mathbf{x}_2$. Throughout this thesis, a Gaussian kernel function is used, $k(\mathbf{x}_1, \mathbf{x}_2) = e^{-\gamma \|\mathbf{x}_1 - \mathbf{x}_2\|^2}$.
- $\gamma$ : Width parameter of the Gaussian kernel function.

## 2.2 Pattern Recognition and Preliminaries

The process of pattern recognition involves an input pattern (also referred to as input datum, observation, input sample, or instance), $\mathbf{x}$, and an output decision, $y$. The input and output of the pattern recognition algorithm are often real values. More specifically the pattern recognition task can be described as in Equation (11).

$$y = f(\mathbf{x}) \tag{11}$$

Unless otherwise stated, $\mathbf{x} \in \mathbb{R}^d$ and $y \in \mathbb{R}$ or $\pm 1$. The pattern recognition algorithm can be considered as a functional mapping, $f : \mathbb{R}^n \to \mathbb{R}$. It is the primary goal of a pattern recognition algorithm to try to produce $y$ as close to the *true value* as possible. The form of the true value will determine whether we are performing binary classification or regression. In the case of binary classification, $f : \mathbb{R}^n \to \{\pm 1\}$. Given an input pattern, the output will determine the class that the input pattern belongs to. The decision values $\{\pm 1\}$ are used due to mathematical convenience as will be discussed later in this chapter. Regression is a more general case where $y \in \mathbb{R}$. In the case of regression, $f$ will be approximating a real-valued function.

The problem of machine learning can be summarized in the following way. Assume that $f$ is not known, and that a set of input patterns, and corresponding output pairs, $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3) \cdots, (\mathbf{x}_m, y_m)\}$ are available. The purpose of a machine learning algorithm is to find an optimal $f$, in an automated fashion. Common vector operations will be made use of when discussing the theories behind support vector machines, namely the dot product ( Equation (12) ) and the euclidean norm ( Equation (13) ).

$$\langle \mathbf{w}, \mathbf{x} \rangle = \sum_{i=1}^{d} \mathbf{w}(i)\,\mathbf{x}(i) \tag{12}$$

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle} \tag{13}$$

$$\|\mathbf{x}\| = \sqrt{\sum_{i=1}^{d} \mathbf{x}(i)^2} \tag{14}$$

## 2.3 Loss and Regularized Risk

For a detailed review of the concepts of this section, readers should refer to [3] and [1]. Given a set of input patterns, $\{(\mathbf{x}_1, y_1), \cdots (\mathbf{x}_m, y_m)\} | \, x, y \in \mathbb{R}^d, \mathbb{R}$ as an example of

a regression application, some fundamental ideas in the area of machine learning can be examined. Figure 6 shows the set of input patterns that can be used to develop the approximating functions, $f_1$ and $f_2$.

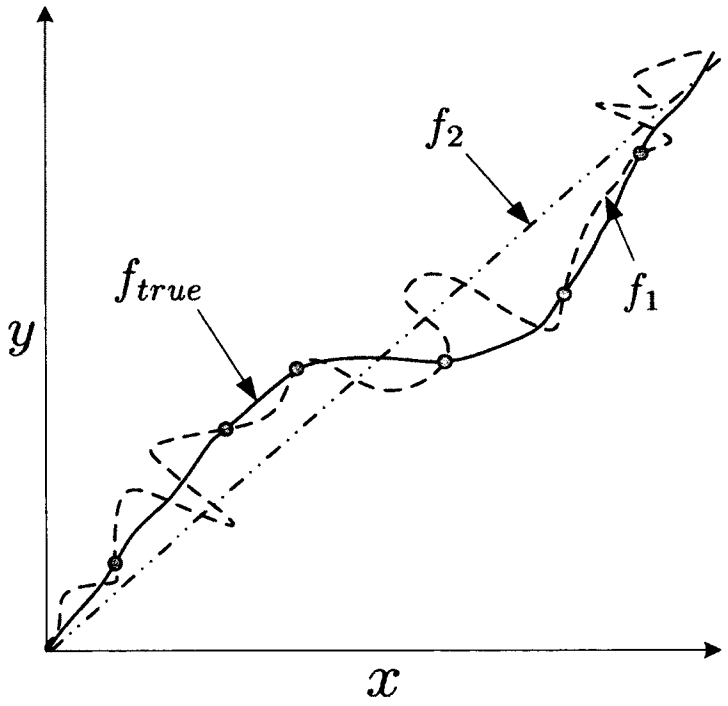


**Figure 6:** An example of two estimating functions where $f_1$ is an example of *over fitting* and $f_2$ is an example of *under fitting* [1]. The true function is illustrated by $f_{true}$.

Assume that $f_1$ and $f_2$ were both determined from the input pattern set. They are examples of *over fitting* and *under fitting*, respectively. The function $f_2$ appears too simple to represent the data and the fact that $f_2$ does not go through any of the samples shows us that there is error. The function $f_1$ has no error with respect to the test patterns, but it seems too complex to accurately represent the underlying functional relationship.

To generalize about the example shown in Figure 6, assume that there is a underlying probability distribution that generated the data, defined as $\mathrm{P}(x, y)$, and there

are $m$ observable samples. The *loss*, or the error associated with the prediction is defined by a *loss* function, $c(x, y, f(x))$. The loss can be interpreted as the amount of error that the function $f(x)$ has compared to the empirical data, $y$, at a specific value of $x$. Equation (15) describes a squared loss function, and Equation (16) [1] [3] describes the $\varepsilon$-insensitive loss function. The *risk*, $R[f]$, is defined in Equation (17) [1] and is the expected value of the loss function across all samples. The square brackets, [ ], are used when operators on a functional are required.

$$c(\mathbf{x}, y, f(\mathbf{x})) = (f(\mathbf{x}) - y)^2 \tag{15}$$

$$c(\mathbf{x}, y, f(\mathbf{x})) = \max\{|f(\mathbf{x}) - y| - \varepsilon, 0\} \tag{16}$$

$$R[f] := \int_{\mathbf{x},y} c(\mathbf{x}, y, f(\mathbf{x}))d\mathrm{P}(\mathbf{x}, y) = \mathrm{E}[c(\mathbf{x}, y, f(\mathbf{x}))] \tag{17}$$

Only the set of patterns are known, and we do not know $d\mathrm{P}(x, y)$. It is assumed that a probability density exists based empirically on the set of patterns. The *empirical risk*, $R_{emp}$, is defined with Equation (18) [1]. As opposed to the expected risk, the empirical risk is amenable to calculation. In this particular case the empirical risk is simply the average loss between $f$ and the set of patterns.

$$R_{emp}[f] := \frac{1}{\mathbf{x}} \sum_{i=1}^{m} c(\mathbf{x}, y, f(\mathbf{x})) \tag{18}$$

$F$ represents an infinite set of possible functions that attempt to describe the set of patterns. Assuming that we can choose any function $f \in F$, finding $f$ that minimizes $R_{emp}$ is insufficient to find the ideal description of the pattern set. The empirical risk gives a method for evaluating the fitness to the test data, but gives no indication on performance to unseen data. The addition of a regularization term, $\Omega[f]$, overcomes

this problem by implicitly defining $F$ as a compact set, and gives an indication of the complexity of $f$. The *regularized risk* [4] [1] functional, $R_{reg}$ with regularization parameter $\lambda$ is defined in Equation (19).

$$R_{reg}[f] = R_{emp}[f] + \lambda\Omega[f] \tag{19}$$

If the expression for the regularized risk were to be minimized, the problem of over fitting would be avoided. Furthermore, if both $R_{emp}[f]$ and $\lambda\Omega[f]$ are convex functions, then a global solution is available.

## 2.4 Kernels

The concept of a kernel function is central to the development of many successful pattern analysis methods including kernel machines. To avoid loss of generality, the input space will now be considered to be $X$ rather than $\mathbb{R}^d$. For instance, one may wish to perform pattern classification or regression on other types of data such as strings, complex numbers, or graphs. In discussing SVMs, one often refers to the *feature space*, $H$. Assume that there is a function, $\Phi()$, that maps from input space to feature space, $\Phi : X \rightarrow H$. The kernel function is defined as $k : X \times X \rightarrow \mathbb{R}$, and can be thought of as a inner product, or dot product, in the feature space described by Equation (20).

$$k(\mathbf{x}_1, \mathbf{x}_2) = \langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2) \rangle \tag{20}$$

The use of kernels is central to the success of kernel machines because kernels allow for the expression of non-linear relationships in the data. Assuming that the kernel function is positive definite and $H$ is a Reproducing Kernel Hilbert Space (RKHS), [40] [1] the kernel function has the reproducing property such that

$\langle f, k(\mathbf{x}, \bullet) \rangle = f(\mathbf{x})$, for $\mathbf{x} \in X$ and $f \in H$ are linear combinations of kernel functions [40]. Expressions 21, 22, and 23 illustrate linear, polynomial (with embedded offset), and the RBF kernel respectively.

$$k(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle \tag{21}$$

$$k(\mathbf{x}, \mathbf{y}) = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^d \tag{22}$$

$$k(\mathbf{x}, \mathbf{y}) = e^{-\gamma \|\mathbf{x}-\mathbf{y}\|^2} \tag{23}$$

The polynomial kernel is parameterized by the degree, $d$, required to be specified by the designer. The RBF kernel is parameterized by the width, $\gamma$, of the Gaussian function. Therefore SVMs that utilize kernels can be applied to specific problems by design of the kernel function, and possible parameterization of the kernel.

The Gram matrix is often used when discussing kernel functions [51]. Given a set of $k$ vectors, the Gram matrix has a dimension of $k$ x $k$ and is defined in Equation (24). It is a matrix representing kernel evaluations between a set of vectors.

$$K_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j) \tag{24}$$

The Gram matrix appears in the QP optimization problem for support vector machines. For use in online processing it can be computed with $O(k)$ computational complexity, and $O(k^2)$ memory complexity.

## 2.5 Support Vector Machines

SVM algorithms compute a decision function, $f$, based on an initial set of data called the *training set.* As described previously in Section 2.3, it is not adequate to search for a minimum in the amount of error that $f$ incurs on the training set. SVMs address this problem by maximizing the *margin* between classes, while minimizing the amount of loss that $f$ exhibits. The most common approach to finding $f$, termed *training*, is through solving a constrained nonlinear optimization problem or nonlinear programming problem. While it is not necessary to be well versed in optimization theory, readers can consult [63] for further reading on constrained optimization using the method of Lagrange, and the Karush-Kuhn-Tucker (KKT) optimality conditions. All kernel machines operate under the principle that a subset of the training set will be used in the evaluation of $f$.

While a detailed derivation of all SVMs will not be reproduced here, it is advantageous to look at the problem of optimal separating *hyperplanes* for the purpose of classification. This problem is also termed the "hard margin" SVM, due to the inability of the method to cope with data that is not linearly separable. Assume that we are given a training set, $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots (\mathbf{x}_m, y_m)\}$, $\mathbf{x} \in \mathbb{R}^2, y \in \{\pm 1\}$. A two dimensional illustration of the training set is given in Figure 7. In the figure, training sample class labels, $y = +1$ and $y = -1$, are represented by circles and squares, respectively.

Equation (25) formulates $f$ as a canonical hyperplane that for every two dimensional input sample, $\mathbf{x}_i$, will calculate the corresponding $y_i$. In Figure 7 the areas where $f(\mathbf{x}_i)$ evaluates to $f(\mathbf{x}) \leq -1$, $-1 \leq f(\mathbf{x}) \leq 1$, and $f(\mathbf{x}) \geq 1$ are indicated by the pink, yellow, and blue regions respectively, while the boundaries where $f(\mathbf{x}) = \pm 1$ and ,$f(\mathbf{x}) = 0$ are indicated by the dashed and solid lines respectively.

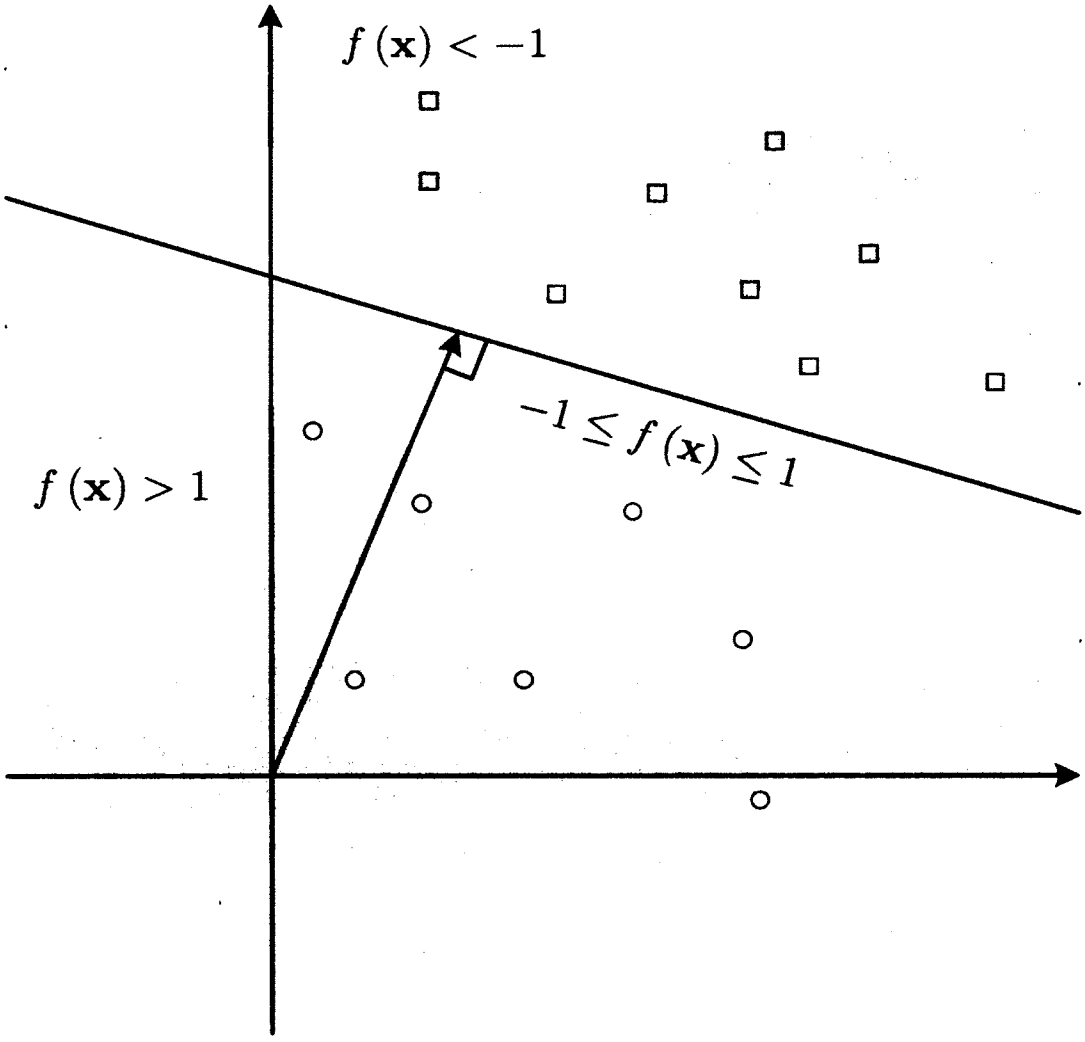


**Figure 7:** Classification using a hyperplanes to separate two classes of data [1].

$$f(\mathbf{x}_i) = y_i = \text{sgn}(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \text{ given that } \mathbf{w}, \mathbf{x}_i \in \mathbb{R}^2, y_i \in \{\pm 1\} \tag{25}$$

The hyperplane that separates the two classes is characterized by $\mathbf{w}$ and $b$. To look to find a $\mathbf{w}$ and $b$ that properly separates the data, the decision function $f$ would exhibit no empirical risk. It is evident from Figure 7 that there are an infinite number of possible decision functions that would correctly separate the data. Minimizing

the empirical risk will be prone to over fitting as described in Section 2.3. The generalization ability of the classifier can be loosely described as the ability of the classifier to correctly identify new, or unseen data. In using the training set as a basis for finding $\mathbf{w}$ and $b$, one can rationalize that it is advantageous to maximize the generalization ability and minimize the empirical risk. Making a modification to Equation (25), with Equation (26), expresses the fact that the optimal hyperplane will compute values in $\mathbb{R}$. Figure 8 introduces the idea of a margin between the two classes in the training set. In this example, the optimal hyperplane will assign $\hat{f}(\mathbf{x}_1) = +1$, and $\hat{f}(\mathbf{x}_2) = -1$. Notice that if one was to find such a hyperplane, only $\mathbf{x}_1$ and $\mathbf{x}_2$ would be needed to find the appropriate values for $\mathbf{w}$ and $b$, other training patterns are not needed.

$$\hat{f}(\mathbf{x}_i) = \langle \mathbf{w}, \mathbf{x}_i \rangle + b \text{ given that } \mathbf{w}, \mathbf{x}_i \in \mathbb{R}^n \tag{26}$$

In this example, the margin of the hyperplane is defined as the projected distance on the normal vector, $\mathbf{w}$, between $\hat{f}(\mathbf{x}_i) = 0$ and $y_i \hat{f}(\mathbf{x}_i) = 1$. As a result, there can be modifications made to the definition of the margin. Multiplying by $y_i$ allows the inclusion of both sides of the class assignment. The margin, $\rho$ is formalized in terms of $\mathbf{w}$ by Equation (27).

$$\begin{aligned} \langle \mathbf{w}, \mathbf{x}_1 \rangle + b &= 1 \\ \langle \mathbf{w}, \mathbf{x}_2 \rangle + b &= -1 \\ \langle \mathbf{w}, \mathbf{x}_1 - \mathbf{x}_2 \rangle &= 2 \\ \left\langle \tfrac{\mathbf{w}}{\|\mathbf{w}\|}, \mathbf{x}_1 - \mathbf{x}_2 \right\rangle &= \tfrac{2}{\|\mathbf{w}\|} \\ \rho &= \tfrac{1}{\|\mathbf{w}\|} \end{aligned} \tag{27}$$

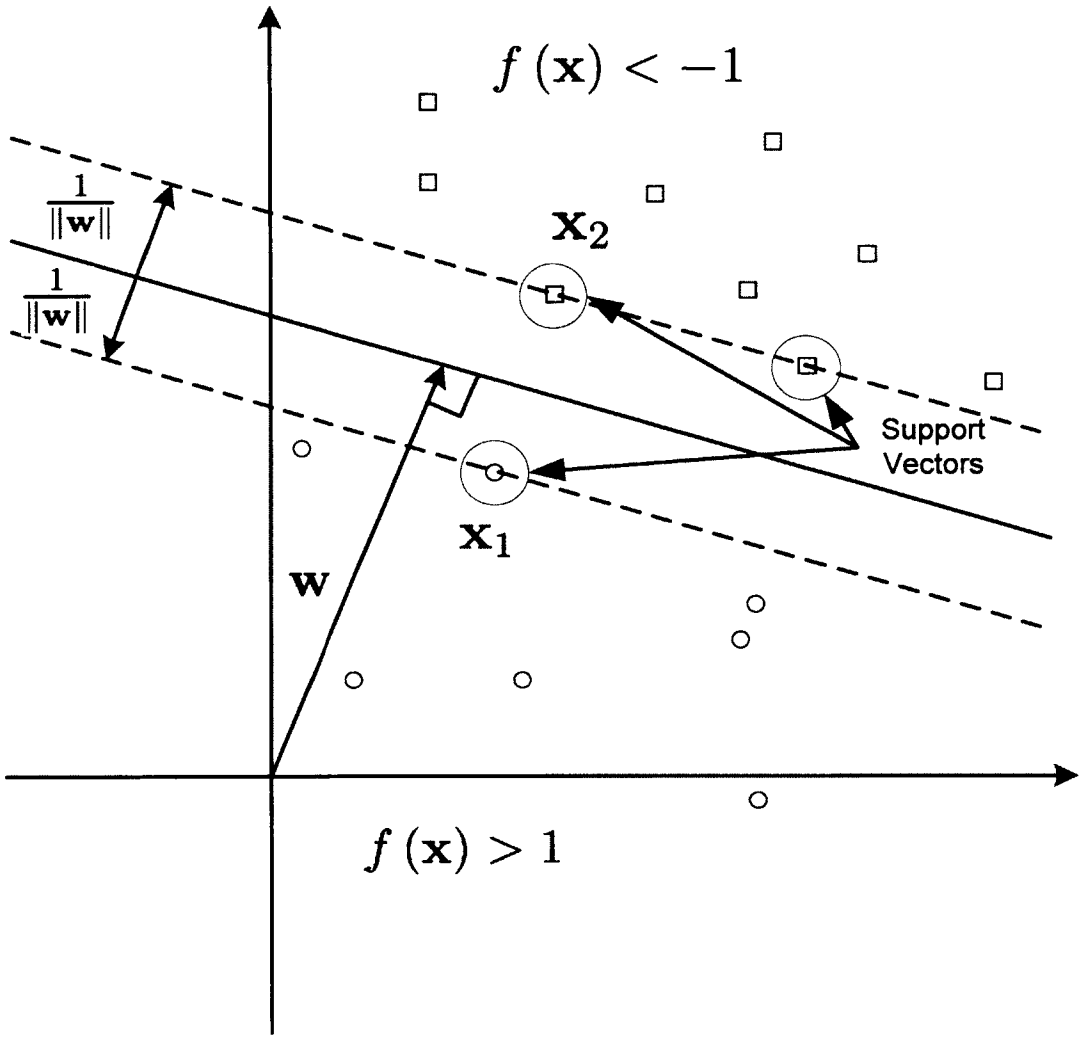


**Figure 8:** Illustration of a maximized margin [1]

With the definition of the margin, one can control the complexity of the function $\hat{f}$. In the case of a canonical hyperplane, controlling the size of the margin effectively finds the "flattest" plane the separates the data while still upholding $\hat{f}(\mathbf{x}_1) = 1$ and $\hat{f}(\mathbf{x}_2) = -1$. The margin can be increased by decreasing $\|\mathbf{w}\|$. The margin gives us a method to enforce regularization to the classification problem.

The general problem of finding a separating hyperplane can be summarized by the

primal optimization problem in Equation (28). The margin is maximized, minimizing the objective function while enforcing the restriction that the hyperplane must separate the data according to class label.

$$\begin{aligned} \underset{\mathbf{w},\mathbf{x}\in\mathbb{R}^n,b\in\mathbb{R}}{\text{minimize}} \quad & \tau(\mathbf{w}) = \tfrac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to:} \quad & y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) \geq 1, \forall i = 1 \ldots m \end{aligned} \tag{28}$$

The form of the optimization problem is easier solved by derivation of the dual representation of Equation (28). First the Lagrangian is introduced in Equation (29), allowing for introduction of the classification constraints into the objective function along with the corresponding Lagrange multipliers, $\alpha_i$. Taking the derivatives of the Lagrangian with respect to $\mathbf{w}$, and $b$ and setting them to 0, as in Equation (30), yields Equation (31) and (32).

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i (y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle + b) - 1) \tag{29}$$

$$\frac{L(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial b} = 0, \frac{L(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial \mathbf{w}} = 0 \tag{30}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \tag{31}$$

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \tag{32}$$

Substituting 31 and 32 into 29 yields the corresponding dual optimization problem in Equation (33). There are three advantages in using the dual for solving the problem. The first advantage is that the expression for the hyperplane is in terms of the training samples, we do not directly calculate $\mathbf{w}$. The classification decision

function, $f(\mathbf{x})$ can now be expressed in 34.

The second advantage is that due to KKT theory, $\alpha_i$ values that are non-zero due to the solution of 33 correspond to the training samples where $y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) = 1, \forall i = 1 \ldots m$. These points lay exactly on the margin and in our previous example, would correspond to $\mathbf{x}_1$ and $\mathbf{x}_2$. These special points represent the *support vectors* of the classifier and are the only points required to evaluate $f(\mathbf{x})$. Therefore given a large set of training samples, the support vectors are kept and the rest can be discarded if no further training samples are introduced.

Finally the third advantage is that 33 is a standard quadratic programming problem, that is convex and has a globally optimal solution. There is a rich body of literature regarding quadratic programming problems in optimization theory and any established algorithm can be used to train the classifier.

$$\begin{aligned} \underset{\alpha \in \mathbb{R}^m}{\text{maximize}} \quad & W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \tfrac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\ \text{subject to:} \quad & \alpha_i \geq 0, \forall i = 1 \ldots m \\ & \sum_{i=1}^{m} \alpha_i y_i = 0 \end{aligned} \tag{33}$$

$$f(\mathbf{x}) = \operatorname{sgn}\left( \sum_{i=1}^{m} \alpha_i y_i \langle \mathbf{x}, \mathbf{x}_i \rangle + b \right) \tag{34}$$

Non-linear kernel functions are used to represent non-linear relationships in the data through the evaluation of the kernel expansion. Section 2.4 introduced the idea of a kernel function representing the implicit dot product in a feature space, $H$. The advantage of using kernels in this case is that $\Phi(\mathbf{x})$ need not be calculated, only the inner product or dot product is important for the formulation of the classifier. Equations (35) and (36) show the described classifier using kernel functions. It is important to use a positive definite kernel in these formulations to ensure that the

optimization problem remains convex and does not introduce local minima. When using kernel functions it is assumed that $\mathbf{w} \in H$ therefore Equation (32) cannot be used to calculate $\mathbf{w}$ because $H$ is an implicit RKHS. The previously described classifier is called a *hard margin* support vector classifier.

$$\begin{aligned} \underset{\boldsymbol{\alpha} \in \mathbb{R}^m}{\text{maximize}} \quad & W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \\ \text{subject to:} \quad & \alpha_i \geq 0, \forall i = 1 \ldots m \\ & \sum_{i=1}^{m} \alpha_i y_i = 0 \end{aligned} \tag{35}$$

$$f(\mathbf{x}) = \operatorname{sgn}\left( \sum_{i=1}^{m} \alpha_i y_i k(\mathbf{x}, \mathbf{x}_i) + b \right) \tag{36}$$

### 2.5.1 C-SVC

The primary problem that existed in the previous section was that we made the assumption that the data was linearly separable. In other words, that there was a separating hyperplane between the two classes of data. The use of a kernel function allows for a possible mapping to a higher dimensional space, $H$, as in Equation (37).

$$y_i \left( \langle \Phi \mathbf{w}, \Phi(\mathbf{x}) \rangle + b \right) \geq 1 \quad \forall i = 1 \ldots m \text{ , given that } \mathbf{w}, \Phi(\mathbf{x}) \in H, b \in \mathbb{R} \tag{37}$$

There must be an allowance for points to lie on the incorrect side of the margin, as shown in Figure 9. The approach used by [64] is to introduce slack variables [1]. Each training sample is given an associated slack variable, $\xi_i \geq 0, \forall i = 1 \ldots m$ to allow for possible error. Although the error is now formalized it is to be avoided when finding an optimal separating hyperplane. The average of all slack variables in the training set is added to the primal objective function in Equation (38) and the constraint is

modified to account for the possibility of error. The parameter $C$ is added so that the influence of the error incurred by the slack variables can be user adjusted. This formulation is referred to as a *soft margin* support vector classifier.

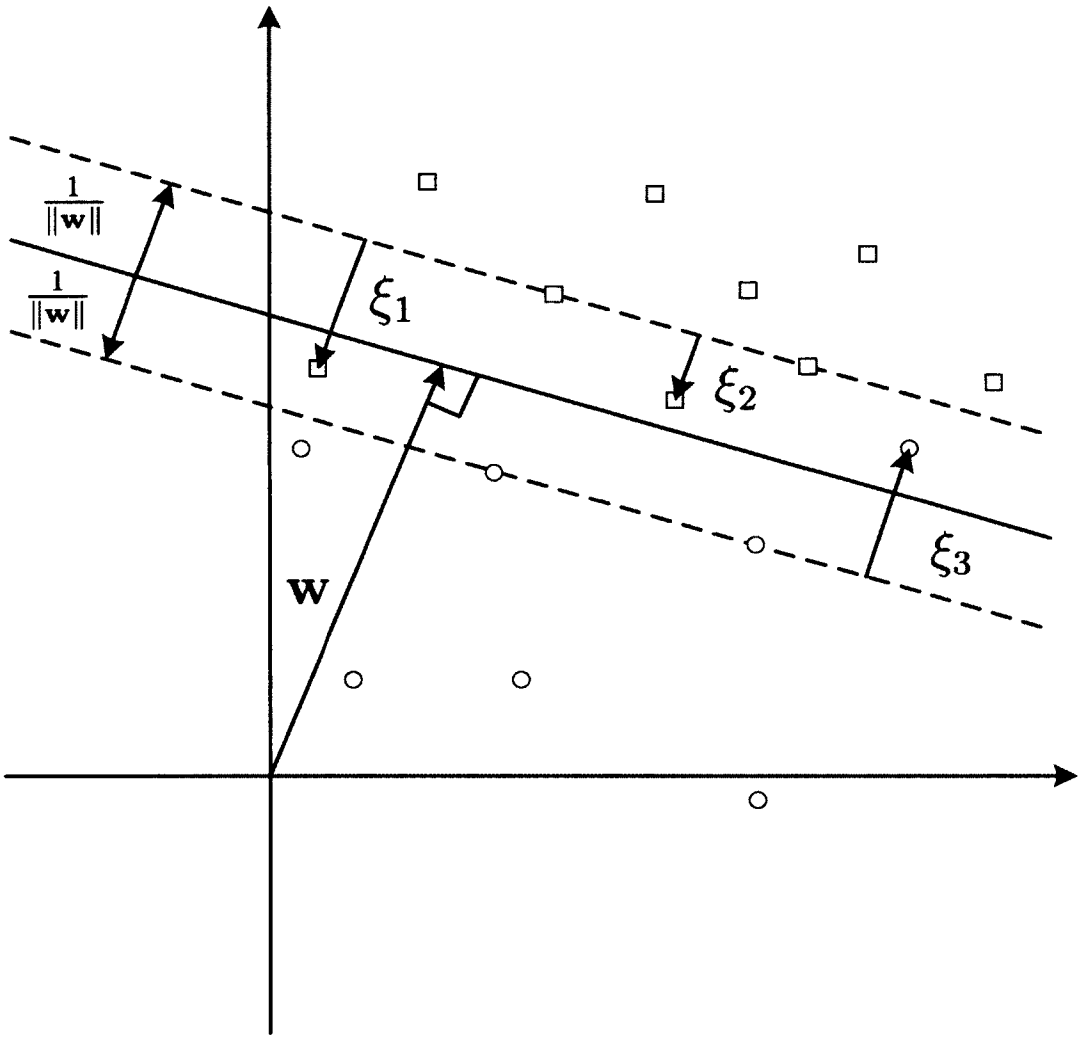


**Figure 9:** Introduction of slack variables [1].

$$\begin{aligned} \underset{\mathbf{w},\Phi(\mathbf{x})\in H,b\in\mathbb{R}}{\text{minimize}} \quad & \tau(\mathbf{w}) = \tfrac{1}{2}\|\mathbf{w}\|^2 + \tfrac{C}{m}\sum_{i=1}^{m}\xi_i \\ \text{subject to:} \quad & y_i\left(\langle \mathbf{w}, \Phi(\mathbf{x}_i)\rangle + b\right) \geq 1 - \xi_i, \forall i = 1\ldots m \\ & \xi_i \geq 0, \forall i = 1\ldots m \end{aligned} \tag{38}$$

A similar dual derivation can be found as in the previous section, the result of which is given in Equation (39). The only difference to the resulting quadratic program is the introduction of a limit to the maximum value that any Lagrange multiplier, $\alpha_i$ can reach. The parameter $C > 0$ has the effect of allowing for the trade-off between minimizing the total error and maximizing the margin of the classifier. Large $C$ brings the classifier closer to the hard margin case, while smaller $C$ allows for more error. The selection of an appropriate parameter is usually done through testing on a set of training and test data, or through leave-one out (LOO) estimates.

$$\begin{aligned} \underset{\boldsymbol{\alpha}\in\mathbb{R}^m}{\text{maximize}} \quad & W(\boldsymbol{\alpha}) = \sum_{i=1}^{m}\alpha_i - \tfrac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \\ \text{subject to:} \quad & 0 \leq \alpha_i \leq \tfrac{C}{m}, \forall i = 1\ldots m \\ & \sum_{i=1}^{m}\alpha_i y_i = 0 \end{aligned} \tag{39}$$

### 2.5.2 SV Novelty Detection

It has been shown in previous sections that a functional representation could be drawn from a set of training data. The training data consisted of an input vector, $\mathbf{x}$, and the observed desired output, $y$. In the case of classification, $y$ represents the class in which the training sample belongs, and for regression $y$ is the real value that the function should take. If we only have $\mathbf{x}$ and not $y$, the problem of binary classification and regression reduce to a single-class classification problem. Other terms used to describe the problem are novelty detection and quantile estimation [62] and [1].

The objective of kernel machine novelty detection is to predict if a specific pattern was produced by an underlying probability distribution. In other words, the kernel machine attempts to detect whether an input pattern is an anomaly based on previous samples. SV novelty detection [62] formulates a hyperplane such that it is separated from the origin by a margin of $\frac{\rho}{\|\mathbf{w}\|}$. Figure 10 illustrates this special situation. It is desirable to have a soft margin in the case of novelty detection to account of possible errors. The standard slack variables, $\xi_i$ are used to allow for samples to lay on the incorrect side of the hyperplane. Rather than use the term "error"; points on the wrong side of the hyperplane can be considered as outliers to the process that is being estimated.

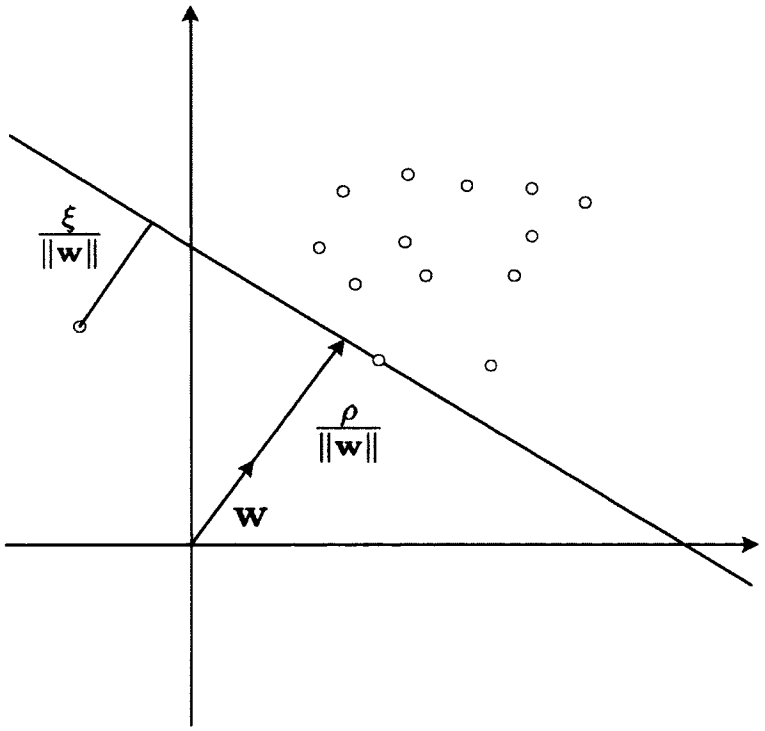


**Figure 10:** Hyperplane separation from the origin for SV novelty detection [1].

The primal optimization problem is given in Equation (40). The use of the parameter $\nu$ is similar to previous formulations in that it provides a lower bound on the

fraction of support vectors that are taken from the training set, and an upper bound on the fraction of outliers existing in the training set. The dual optimization problem is given in Equation (41).

$$\begin{aligned} \underset{\mathbf{w},\Phi(\mathbf{x})\in H,\rho\in\mathbb{R}}{\text{minimize}} \quad & \tau(\mathbf{w}) = \tfrac{1}{2}\|\mathbf{w}\|^2 + \tfrac{1}{\upsilon m}\sum_{i=1}^{m}(\xi_i - \rho) \\ \text{subject to:} \quad & \langle \mathbf{w}, \Phi(\mathbf{x}_i)\rangle \geq \rho - \xi_i, \forall i = 1\ldots m \\ & \xi_i \geq 0, \forall i = 1\ldots m \end{aligned} \tag{40}$$

$$\begin{aligned} \underset{\boldsymbol{\alpha}\in\mathbb{R}^m}{\text{minimize}} \quad & W(\boldsymbol{\alpha}) = \tfrac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j k(\mathbf{x}_i,\mathbf{x}_j) \\ \text{subject to:} \quad & 0 \leq \alpha_i \leq \tfrac{1}{\upsilon m}, \forall i = 1\ldots m \\ & \sum_{i=1}^{m}\alpha_i = 1 \end{aligned} \tag{41}$$

The decision function for the novelty detector is given in Equation (42). Instead of the traditional bias, $b$, the margin scale factor $\rho$ is used. The details in computing an optimal value for $\rho$ will be addressed when discussing current training methods for support vector based novelty detection.

$$f(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^{m}\alpha_i k(\mathbf{x}_i,\mathbf{x}) - \rho\right) \tag{42}$$

The discussion of the various SVM algorithms has involved solving the dual QP problem with $m$ variables. When the SVM is trained with $m$ input samples, some of QP variables become zero and are no longer needed in the formulation of the decision function. At this point the variable $m$ is still used but now represents the number of non-zero Lagrange multipliers and their associated input patterns. In other words, $m$ represents the number of input samples before training, and then represents the number of support vectors. The execution efficiency of a SVM is directly related to

the number of support vectors.

## 2.6 Naive Online $R_{reg}$ Minimization Algorithm

The Naive Online $R_{reg}$ Minimization Algorithm (NORMA) operates by training a kernel machine using stochastic gradient descent such that the regularized risk functional becomes smaller with each time step. Equation (43) describes the regularized risk functional that the NORMA uses when training a kernel machine. The empirical risk is defined as in the previous section. The regularization term is parameterized such that the amount of regularization can be controlled by setting the parameter $\lambda$.

$$R_{reg}[f] = R_{emp}[f] + \frac{\lambda}{2}\|f\|_H^2 \tag{43}$$

The NORMA represents a wide family of kernel machine algorithms. Equations (45) and (46) describe the update rules for the current expansion term weight at time $t$, $\alpha_t$, all previous expansion term weights, $\alpha_i$, and scalar bias, $b$ for a binary classifier kernel machine and a support vector novelty detector respectively. Equation (44) is an intermediate switch variable for assignment of $\alpha_t$. The update equations for all previous expansion weights ($\alpha_i$'s) reveal that the expansion term weights decay as time progresses. The use of stochastic gradient descent using linear loss enforces regularization in RKHS through time-based weight decay. An interested reader should consult [40] for full explanation of the NORMA.

$$\sigma_t = \begin{cases} 1 & f(\mathbf{x}_t) < \rho \\ 0 & otherwise \end{cases} \tag{44}$$

$$(\alpha_i, \alpha_t, b) \leftarrow ((1-\eta\lambda)\alpha_i, \eta\sigma_t y_t, b + \eta\sigma_t y_t) \tag{45}$$

$$(\alpha_i, \alpha_t, b) \leftarrow \begin{cases} ((1-\eta)\,\alpha_i, \eta, \rho + \eta\,(1-\nu)) & f(\mathbf{x}_t) < \rho \\ ((1-\eta)\,\alpha_i, 0, \rho - \eta\nu) & otherwise \end{cases} \tag{46}$$

Equation (46) requires a slight modification due to an arithmetic error. Taking the derivative of the hinge loss function and applying stochastic gradient descent yields the update for a support vector novelty detector in Equation (47).

$$(\alpha_i, \alpha_t, b) \leftarrow \begin{cases} ((1-\eta)\,\alpha_i, \eta, \rho - \eta\,(1-\nu)) & f(x) < \rho \\ ((1-\eta)\,\alpha_i, 0, \rho + \eta\nu) & otherwise \end{cases} \tag{47}$$

The update procedure of the NORMA requires the kernel expansion be calculated with every time step for the current input only. The computational complexity of training a kernel machine using the NORMA is $O(t)$ where the number of time steps equals the number of support vectors stored in the kernel machine ($t = m$).

## 2.7 Online Kernel Methods for Regression Tasks

Kernel machines use multidimensional inputs for regression tasks and it is often required to use the technique of *time embedding*. A consistent method for time embedding is used in this thesis. The first step is to define the input dimension, $n$. The input dimension will effect the computational complexity of the kernel evaluation. The second step is to window the data with a window size of $n + 1$. At any time $t$, $\mathbf{x}_t$ and $y_t$ are defined in Equation (48) and Equation (49) respectively. All cases of time embedding within this work use $t \geq n + 1$. Figure 11 illustrates the use of time embedding on the time series $\mathbf{p}_t$.

$$\mathbf{x}_t = \{\mathbf{p}_t\,(t-n) \cdots \mathbf{p}_t\,(t-1)\} \tag{48}$$

$$y_t = \mathbf{p}_t(t) \tag{49}$$

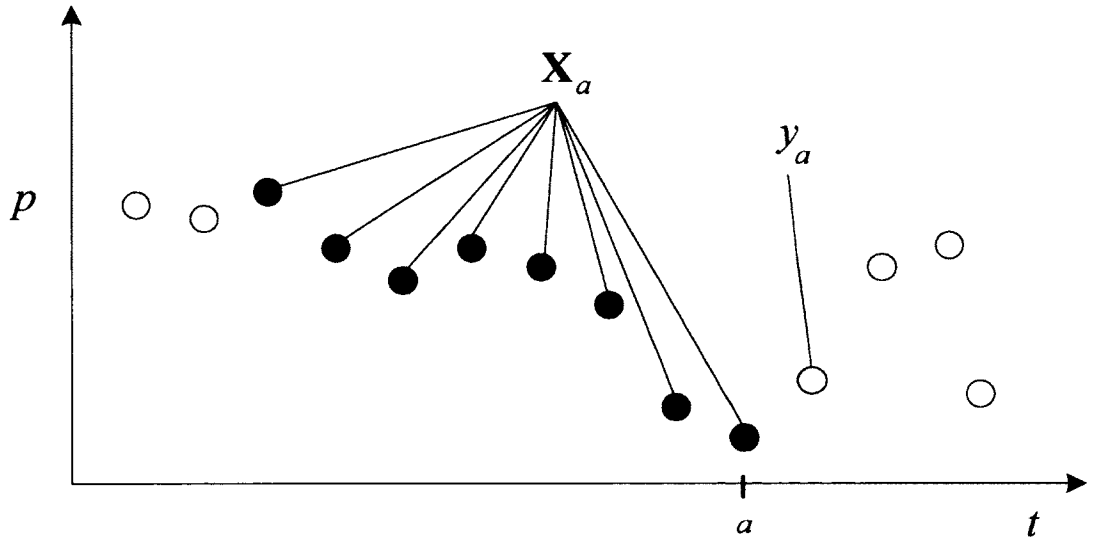


**Figure 11:** An illustration of the time embedding procedure used in this thesis.

## 2.8 Kernel Least Mean Square Algorithm

The kernel least mean square (KLMS) adaptive filter algorithm makes use of the well known least mean square (LMS) adaptive filter to provide kernel machine algorithm that can be used for regression and function approximation. The KLMS algorithm takes advantage of inner products in RKHS to build a stable non-linear filter in the kernel machine's input space. An interested reader can consult the works of Liu et al. [51, 53] for more details on the KLMS algorithm. Algorithm 1 describes the training procedure for KLMS. The KLMS algorithm runs with infinite memory, but later in Chapter 5 it will become evident that KLMS can be unstable under certain conditions when using a finite size buffer for storing the support vectors.

**Algorithm 1** The KLMS Algorithm [51].

Input: Step Size $\eta$, Kernel Function, $k()$
Initialize: Support Vector Buffer $X \leftarrow \emptyset$
**while** $t < \infty$ **do**
  From Input, get $\{\mathbf{x}_t, y_t\}$
  $\hat{y}_{t-1} = \sum_{i=1}^{t-1} \alpha(i) k(\mathbf{x}_t, \mathbf{x}_i)$
  $\alpha(t) \leftarrow \eta(y_t - \hat{y}_{t-1})$
  $X \leftarrow X \cup \mathbf{x}_t$
**end while**

# Chapter 3

# Stochastic Subset Selection

Online learning needs to be adaptive to changes in the concepts the learner is trying to predict. The process of changing concepts over time is called *concept drift* and many approaches can be used in dealing with or detecting concept drift. One approach is using an ensemble of multiple experts, trained at different points in time, to detect and adapt to concept drift [60, 65–68]. The use of an ensemble of classifiers has an advantage that classifiers can be removed from the ensemble if their learned concepts are no longer valid.

A different approach can be used where a window of retained knowledge is kept and the size of the window can be fixed or adjusted depending on how much concept drift has occurred [57, 69–71]. SVMs and other kernel machines are good candidates for this approach because the samples in the data stream serve as a basis for the classification or regression hypothesis. Under certain types of concept changes, it is most effective to clear all the support vectors from the kernel machine [72].

The method proposed in this chapter allows a mixing of these two approaches. Only one classifier is used as well as a stochastic indexing technique that makes it more probable to pick recent data to form a decision hypothesis. At the same time the computational complexity of evaluating the kernel expansion is reduced because only a subset of kernel machine vectors are used for evaluation and training.

## 3.1 Chapter Outline

The first section of this chapter describes a computationally efficient method for the generation of index values. These index values will be used for choosing the kernel machine vectors that will make up the subset for the kernel expansion evaluation. The second section will describe the novel Stochastic Subset Selection (S3) algorithm. The third and fourth sections describe the effect of induced stochastic noise in the classifier output as an effect of the S3 algorithm and compensation of this noise respectively. The fifth section gives experimental results on simulated and benchmark datasets.

## 3.2 Fast Variate Generation

There are $m$ support vectors to contribute to the classifier output, and the $m$ support vectors fill the available buffer space. Since a subset of all available support vectors need to be selected, the ordered set of SVs can be accessed by their respective index values. The SVs are indexed from 0 to $m-1$, 0 being the most recently included support vector, and the $m-1$th SV being the oldest support vector.

The ability of a kernel machine to adapt to changes in its environment depends on how much old, non-relevant information is stored in it's memory. The S3 algorithm uses random variates when choosing buffer index values for the kernel expansion evaluation and to allow for adaption to a changing environment, the variate generation is based on a modified exponential probability density function (pdf). The exponential pdf allows the system designer to specify how likely recent data in the buffer will be used in the kernel expansion.

An inverse transform sampling on the interval $[0 \ldots 1)$ of the cumulative distribution function (cdf) of the modified exponential distribution is used to generate a single variate. The pdf, $g(x)$, and cdf, $G(x)$, used in the S3 algorithm are given in

equations 50 and 51 respectively.

$$g(x) = \begin{cases} \frac{a}{1-e^{-a}} e^{-ax} & x \in [0, 1.0) \\ 0 & \text{otherwise} \end{cases} \tag{50}$$

$$G(x) = \int g(x)\, dx = \begin{cases} \frac{1-e^{-ax}}{1-e^{-a}} & x \in [0, 1.0) \\ 0 & \text{otherwise} \end{cases} \tag{51}$$

Figure 12 illustrates the effect of the parameter $a$ on the cdf. The functions $g(x)$, and $G(x)$ are referred to as modified exponential distributions due to the normalization of $G(x)$ on the range $x \in [0, 1.0)$. The cdf, $G(x)$ will evaluate to 1.0 as $x \to 1.0$. A biased random number generator using $X$ and its associated $G(x)$ can be used to favor smaller values of $x$ or generate uniform values on the range $x \in [0, 1.0)$. As $a \to 0.0$ the random number generator will generate uniform values on the range $x \in [0, 1.0)$, as $a$ gets larger, the random number generator will be biased towards smaller values on the range $x \in [0, 1.0)$.

To generate a random variate, we use a uniformly distributed random variable, $X$ on the range $[0 \ldots 1)$. The random variate is calculated from Equation (52).

$$i = \left\lfloor \frac{-\ln(1 - x(1 - e^{-a}))}{a} \times m \right\rfloor \tag{52}$$

Assuming the generation of the random variable $X$ is an elementary operation operating in $O(1)$ time, the variate generation also operates in $O(1)$ time.

## 3.3 The Stochastic Subset Selection Algorithm

One of the the main contributions of the work presented in this thesis is the Stochastic Subset Selection (S3) algorithm. The fast variate generation method given by

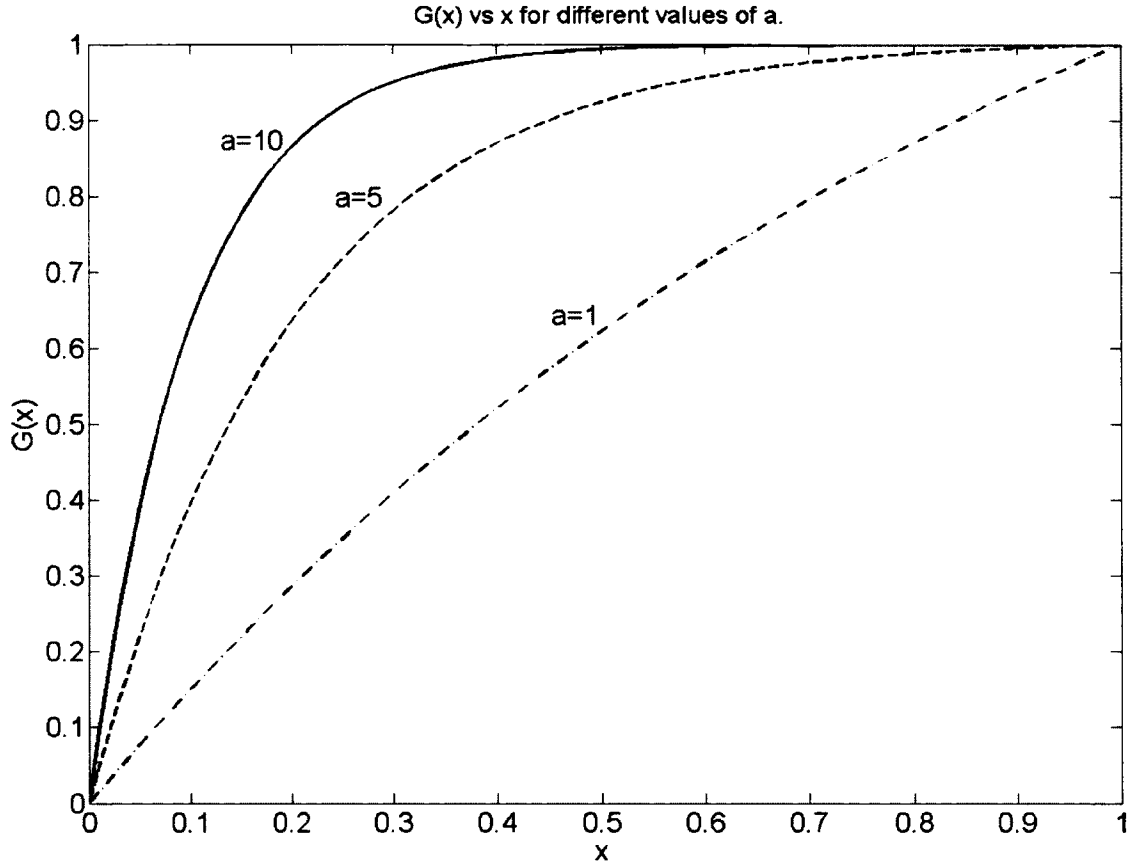


**Figure 12:** Examples of how the shape of the cdf, $G(x)$, changes with different values of the rate parameter, $a$. The exponential distribution has been normalized such that the probability of $x \leq 1.0$ will always be 1.0.

Equation (52) is used to generate index values for the buffer of support vectors. All support vectors are assumed to reside in a FILO buffer of size $m$, and are indexed from 0 to $m-1$. The S3 algorithm makes use of two sets while selecting a subset of support vectors from the buffer. The first set, $M$, is initially an ordered set of integers ($M = [0, 1, \cdots, m-2, m-1]$). The second set, $B$, is initially empty ($B = \emptyset$) and will contain the selected kernel terms for the expansion evaluation. As in the previous section, the variable $x$ is sampled from a uniformly distributed random variable, $X$. The number of elements in the subset produced by the S3 algorithm is user defined by the parameter $\beta \in [0.0, 1.0]$.

**Algorithm 2** The S3 Algorithm

Initialize:

- $M \leftarrow [0, 1, \cdots, m-2, m-1]$
- $B \leftarrow \emptyset$
- $n \leftarrow m$

**while** $|B| \leq \lfloor \beta m \rfloor$ **do**
$x \leftarrow X$
$i = \left\lfloor \frac{-\ln\left(1-x\left(1-e^{-\alpha}\right)\right)}{\lambda} \times n \right\rfloor$
$B \leftarrow B \cup M[i]$
$M \leftarrow M \backslash M[i]$
$n \leftarrow n - 1$
**end while**
Return $\hat{f}(\bullet) = \sum_{j \in B} \boldsymbol{\alpha}(j) k(\bullet, \mathbf{x}_j)$

The S3 algorithm uses $m$ elementary operations during initialization, and $\lfloor \beta m \rfloor$ elementary operations inside the while loop. The S3 algorithm has $O(m)$ time complexity, but in practice it will perform faster than using all kernel expansion terms because the computational complexity of the kernel function evaluation dominates the computational effort of expansion evaluation. A more detailed analysis of the computational efficiency will be given in Section 3.6.

## 3.4 Analysis of Noise

The S3 algorithm uses a stochastic subset of the stored SVs for kernel expansion evaluation, and therefore represents an approximation of the classification function the kernel machine is representing. Equation (53) describes the amount of approximation error, that the S3 algorithm introduces into the kernel evaluation. The approximation error is referred here as noise, because it varies as a stochastic process that resembles additive noise in the classifier output. The set $Q$ contains the buffer index values that

are not included in the subset selection, therefore $|Q| = \lfloor (1-\beta)\, m \rfloor$. The approximation error, $\varepsilon_\beta(x)$, is both a function of the chosen subset factor, $\beta$, and the input pattern being classified.

$$\begin{aligned} \varepsilon_\beta(\mathbf{x}_t) &= f(\mathbf{x}_t) - \hat{f}(\mathbf{x}_t) \\ &= \left( \sum_{i \in M} \alpha_i k(\mathbf{x}_t, \mathbf{x}_i) \right) \\ &\quad - \left( \sum_{j \in B} \alpha_j k(\mathbf{x}_t, \mathbf{x}_j) \right) \\ &= \sum_{l \in Q} \boldsymbol{\alpha}(l)\, k(\mathbf{x}_t, \mathbf{x}_l) \end{aligned} \tag{53}$$

Equation (54), $\|\hat{\varepsilon}_\beta(\mathbf{x})\|$, approximates the magnitude of the noise in the classifier output. The random variable $A_K$ is used to represent all $\boldsymbol{\alpha}(l)\, k(\mathbf{x}, \mathbf{x}_l)$ evaluations that are indexed by the set $Q$. The underlying probability distribution for $A_K$ is unknown, but it is dependent on the training method used to determine $\boldsymbol{\alpha}(l)$, the type of kernel function used, $k(\mathbf{x}, \mathbf{x}_l)$, and the nature of the input data, $\mathbf{x}$.

$$\|\hat{\varepsilon}_\beta(\mathbf{x}_t)\| = \lfloor (1-\beta)\, m \rfloor\, E[\|A_K\|] \tag{54}$$

The summation term is removed in Equation (54) and the expected value operator is used along with a scaling factor that is based on $\beta$. As the value of $\beta$ becomes lower, the number of index values in $Q$ grows larger, in effect leaving more kernel expansion terms out of the evaluation. As $Q$ becomes larger the magnitude of the noise also becomes larger. The effects of the noise are presented in Section 3.7.

## 3.5 Noise Compensation

The S3 algorithm provides an efficient method for evaluating the decision function of the online kernel machine classifier, but it also creates noise in the resulting decision

function. To compensate for the noise, two moving average filters are utilized on the output of the kernel machine. Figure 13 illustrates how the S3 algorithm can be used in a machine learning system in conjunction with NORMA.

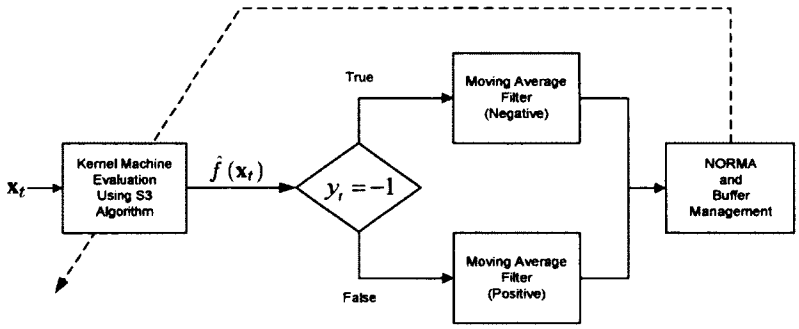


**Figure 13:** A system diagram showing the parallel output filters and feedback that occurs when learning is being performed. The output of the kernel machine at time $t$, using the S3 algorithm is $\hat{f}(\mathbf{x}_t)$. The input label at time $t$ is $y_t$.

The parallel filters prevent the noise from interfering with the learning mechanism provided by the NORMA. Equation 54 illustrates that the nature of the noise is dependent on various system parameters and the input ($\beta$, $k()$, $\boldsymbol{\alpha}$, and $\mathbf{x}$). How the filters are designed is dependent on how the system designer makes use of the kernel machine and the nature of the input. Equation (54) can give some guidance to the amount of noise added to the output of the SVM.

When using the S3 algorithm in a learning scenario, such as in Figure 13, $y_i$ for each input pattern is available, and is used for determining in which output filter to place the noisy output. The output of the filter is then used as the kernel expansion evaluation used in NORMA. The use of the S3 algorithm in this way requires no modification of the learning algorithm.

Figure 14 illustrates a scenario that uses the kernel machine for classification only (no additional data will be added to the kernel expansion). The classification system benefits from the efficiency of the S3 algorithm, but must employ $\hat{f}(x)$ for deciding which filter to use.

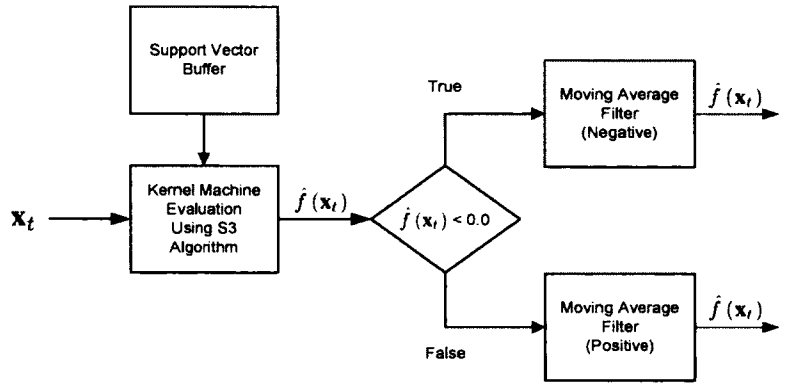


**Figure 14:** Evaluation of the kernel expansion still makes use of the parallel output filters. In this setting, the stochastic estimate, $\hat{f}(\mathbf{x}_t)$ exhibits noise.

The primary drawback in Figure 14 is that the state of the filters will change as inputs are evaluated. The noise is non-linear, and the output filters will drift away from their respective classification labels unless more labeled data is introduced. It is also possible to reset the output filters by retraining on the support vectors.

## 3.6 Computational Complexity

The investigation of kernel machine execution time is usually based on the number of kernel function evaluations involved when computing the kernel expansion. Kernel function evaluation involves floating point operations on vectors of dimension $d$. Without the loss of generality the RBF kernel function, Equation (23), will be considered when discussing the computational complexity of kernel function evaluation.

How much computational benefit a kernel machine receives when using the S3 algorithm becomes evident when directly comparing the number of kernel evaluations. Equation (55) describes the speedup exhibited when using the S3 algorithm. Assuming the kernel machine buffer is full and has total capacity of $m$, the speedup received when computing the kernel expansion is inversely proportional to the subset factor, $\beta$. The constant $c$ in Equation (55) is the extra computational effort in generating

$\lfloor \beta m \rfloor$ random indicies, which scales linearly with buffer size.

$$\text{Speedup} = \frac{m}{\beta m + \beta c m} = \frac{1}{\beta (1 + c)} \tag{55}$$

Figure 15 gives the ratio of the time taken for a single kernel function evaluation and the time taken for a single variate calculation vs the input dimension $d$. The y-axis in Figure 15 shows that kernel evaluation takes time that is multiple orders of magnitude longer than variate generation. This data presented in this plot was implemented in Matlab on a Intel Core i7-2670QM.

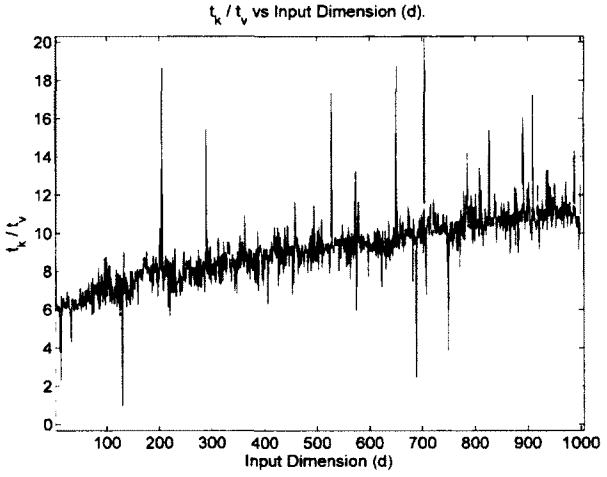


**Figure 15:** $t_k$ and $t_v$ represent the time taken for kernel function evaluation and fast variate generation respectively.

The output filtering used in this chapter is a simple moving average (which has low-pass characteristics). Unless otherwise stated, the length of the moving average is five time steps, which has a computational cost similar to that of the fast-variate generation. Both the time taken for fast variate generation and output filtering result in a low value for $c$ in Equation (55). The choice of output filtering is problem dependent, and requires an analysis of the trade off between error achieved by the classifier and computational efficiency of the classifier function. The $\beta$ parameter

must be chosen in a similar fashion and is optimized off-line in Section 3.7.

## 3.7 Experimental Results

The performance of the S3 algorithm was tested in three different settings. The first setting involved the classification of low dimensional input patterns. The second setting involved the classification of handwritten digit recognition in a high dimensional input space using the popular MNIST [59] dataset. Finally, the third setting involves a large benchmark dataset that was introduced in the work of Street and Kim [60]. In each setting, where appropriate, various aspects of the S3 algorithm were verified including the ability to track changes in input, execution efficiency, effect of noise, and learning ability.

Rosenblatt's perceptron algorithm [43] is the first, and the most simplistic example of an online machine learning algorithm. The perceptron algorithm can be extended to use kernel functions, creating a simple kernel machine for classification [2]. The training procedure for the kernel perceptron is only error driven, and does not take advantage of a large margin. The S3 algorithm was combined with NORMA (S3+NORMA) and was validated against both NORMA and the kernel perceptron.

In all cases, a support vector is only added when there is an error (kernel perceptron) or a margin error (NORMA). When the buffer size is reached, older support vectors are truncated from the buffer. When calculating random variates, the lower the index value for the buffer index, the more recent the support vector. Since the S3 algorithm is stochastic in nature, all classifier accuracies are reported by performing 10 trials of the experiment. The average, maximum, and minimum are then tabulated.

### 3.7.1 Classification Results of Low Dimensional Synthetic Data

One thousand two dimensional data points were generated from two Gaussian random variables both with the same parameters, $\Phi(\mu = 0.0, \sigma = 1.0)$. The associated labels, $y_i$, for each sample were determined by which quadrant on the Cartesian plane the point fell into. The label $y_i = +1$ when $(x(1) \geq 0.0 \wedge x(2) \geq 0.0) \vee (x(1) < 0.0 \wedge x(2) < 0.0)$, and $y_i = -1$ otherwise. To introduce switching concepts into the data, the label assignment was reversed every 120 time steps. Finally two types of drifting were introduced in the label assignment, slow drifting and fast drifting. In the slow drifting case, the label boundaries were rotated clockwise about the Cartesian plane at a rate of 0.72 degrees per time step. In the fast drifting case, the label boundaries were rotated clockwise about the Cartesian plane at a rate of 5.76 degrees per time step. Figure 16 illustrates the label assignments in the stationary, switching and drifting cases.

This experiment is a toy classification problem that embodies both stationary data and non-stationary data. In the stationary case it is impossible to accurately classify the input data with a linear classifier. A real world example that is similar to the non-stationary case of drifting would be classification of a two dimensional moving target over time. In the switching case it could be the classification of data the exhibits a step discontinuity such as classification of a physical object the strikes a solid barrier.

The same Gaussian kernel was used for all experiments involving the above described datasets. Although kernel machines are a non-parametric tool for regression and classification, depending on the kernel used, an optimal kernel parameter ($\gamma$) must be calculated. The NORMA uses *stochastic gradient descent* of the regularized empirical risk function [40] to learn a classification boundary, and an appropriate

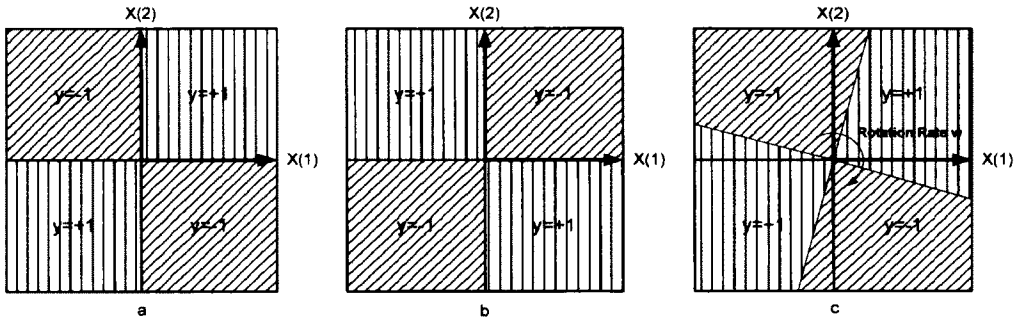


**Figure 16:** (a): Stationary labeling, 1000 samples. (b): Switched labeling every 120 samples in sequence. (c): Slow drifted labeling, labeling regions rotated at a rate of $w = 0.72$ degrees per sample. Fast drifted labeling, labeling regions rotated at a rate of $w = 5.76$ degrees per sample.

learning rate, $\eta$, must also be calculated. Two-fold cross-validation and a particle swarm optimization (PSO) [73] algorithm is used to find the best values for $\gamma$ and $\eta$ on the stationary dataset. Cross validation techniques are statistically sound methods for evaluating the performance of a predictive model while guarding against over fitting of the training data, but must be used on stationary data. When the appropriate parameters for NORMA are found, the S3 algorithm is tested on the switching, slow and fast drifting datasets. All experiments used a fixed buffer size of 60 samples. A finite size buffer was used so that the maximum computational complexity of the kernel machines could be limited. Limited buffer size is a realistic condition for online systems. The maximum number of support vectors present is limited to 60, but SVs are only added when there is a margin violation. The buffer is circular in nature, therefore once the buffer is full the oldest support vectors will be replaced with newly arriving data.

The empirical risk (using a linear loss function) of the classifier on the training set is used during our evaluations rather than the error rate, because it gives a smoother objective function to optimize over. Equations (56) and (57) define the linear loss and empirical risk functions respectively.

$$l\left(f\left(\mathbf{x}_i\right), y_i\right) = \begin{cases} 1 - f\left(\mathbf{x}_i\right) & y_i = 1 \wedge f\left(\mathbf{x}_i\right) < 1 \\ 1 + f\left(\mathbf{x}_i\right) & y_i = -1 \wedge f\left(\mathbf{x}_i\right) > -1 \\ 0 & otherwise \end{cases} \tag{56}$$

$$R_{emp}\left(f, X, Y\right) = \sum_{i=1}^{t} l\left(f\left(\mathbf{x}_i\right), y_i\right) \tag{57}$$

Figure 17 illustrates the use of the PSO algorithm using 2-fold cross validation. Caption (a) illustrates the separation between +1 and -1 classes. In caption (b) of Figure 17, the PSO algorithm finds the optimal $\gamma$ and $\eta$ values when optimizing the empirical risk, $R_{emp}\left(f, X, Y\right)$ on the two-dimensional dataset. After optimal kernel and learning rate parameters are found, the S3 algorithm is run varying $\beta$ and $a$ in a grid pattern, which is illustrated in caption (c). The dark contour line illustrated in the plot of caption (c) represents the empirical risk associated with a NORMA classifier learning the classification of positive and negative labels. For stationary data the S3 enhancement has little effect. No matter which values we choose for $\beta$ or $a$, the loss is greater than using all the vectors stored in the kernel machine. This result makes sense because there are no changes in either the input features, $X$, or the concept generating the labels, $Y$.

Captions (d), (e) and (f) of Figure 17 reveal some significant information regarding the performance of the classifier using the S3 algorithm. In all three captions the area above and to the right of dark contour line represent areas exhibiting smaller amounts of empirical risk than if the NORMA algorithm was used directly, without the stochastic sub-sampling of support vectors that the S3 algorithm offers. Caption (d) illustrates the performance of the S3 algorithm in the presence of switching labels. Every 120 samples the labels swap positions as described in caption (a) in Figure 16. The parameters $\beta = 0.57$ and $a = 17.4$ were chosen visually from the plot minimize the

empirical risk verses the number of kernel terms used in the expansion of the classifier. In a similar fashion, captions (e) and (f) in Figure 17 illustrate the parameter choices for the slow and fast drifting concept changes.

Figure 18 shows the execution performance of 4 different kernel classifiers. The NORMA and Kernel Perceptron classifiers exhibit linear execution profiles as their buffers grow. In the case of both NORMA+S3 and NORMA+S3 (without output filtering), the execution time is limited directly by the number of vectors used to make up the kernel expansion. Since the memory buffer is circular, the execution time becomes deterministic when it is filled. The gain in computational efficiency is scaled by $\beta$. A computer vision application is an example where $\beta$ can be tuned to meet real-time requirements.

The bottom plot in Figure 18 shows the linear growth of the kernel machine depending on the subset selection factor, $\beta$. In the bottom plot in Figure 18 it is evident that the use of moving average filters has a negligible effect on the kernel machine's execution time.

Table 4 describes the classification accuracy results for the 2D stationary and switching concept drift experiments. Different values of the NORMA regularization parameter, $\lambda$, were used so that the effect of regularization of the S3 algorithm and NORMA can be compared. The stationary dataset shows the the regularization effect the S3 algorithm has on the classifier accuracy is the same as the NORMA using $\lambda = 0.1$. The S3 algorithm will evaluate the classifier output more efficiently because only a subset of support vectors are being used. In the presence of switching in the input data the S3 algorithm performs slightly better than the NORMA, but will be less computationally expensive. Finally output filtering has a positive effect on accuracy when the data is stationary, and a negative effect on accuracy when the data exhibits concept drift.

Table 5 describes the experimental results for the 2D drifting data experiments.

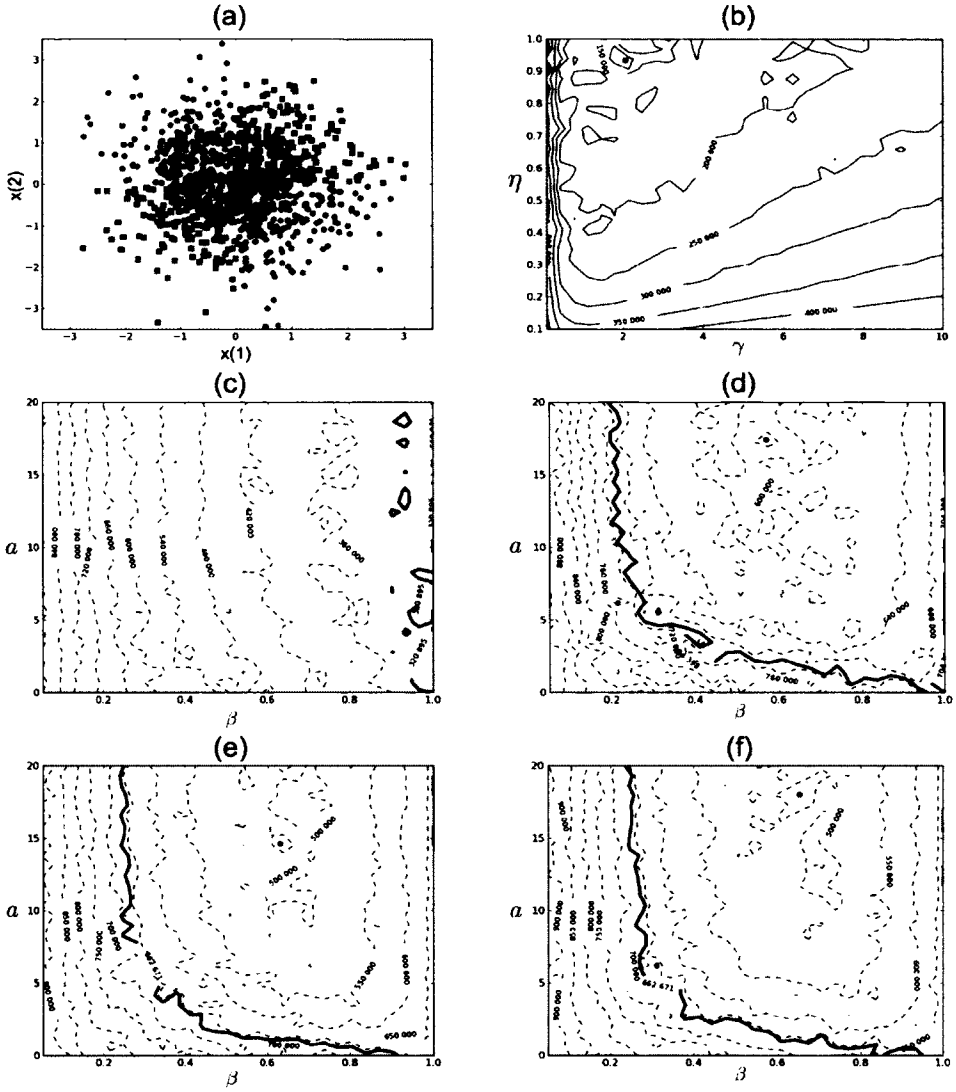


**Figure 17:** (a): Two dimensional scatter plot of the 1000 point stationary dataset. Blue squares are +1 labeled samples, and red circles are -1 labeled samples. (b): A contour plot of $R_{emp}(f, X, Y) = \sum_{i=1}^{t} l(f(\mathbf{x}_i), y_i)$ using 2-fold cross validation. Optimum found at $\gamma = 2.06$, $\eta = 0.935$. (c): A contour plot $R_{emp}(f, X, Y) = \sum_{i=1}^{t} l(f(\mathbf{x}_i), y_i)$ using NORMA+S3 (no output filter) on the 1000 sample stationary dataset. Dark contour line represents empirical risk using NORMA without the S3 algorithm. (d): S3 operating point chosen at $\beta = 0.57$, $a = 17.4$. (e): S3 operating point chosen at $\beta = 0.63$, $a = 14.6$. (f): S3 operating point chosen at $\beta = 0.65$, $a = 18.0$.

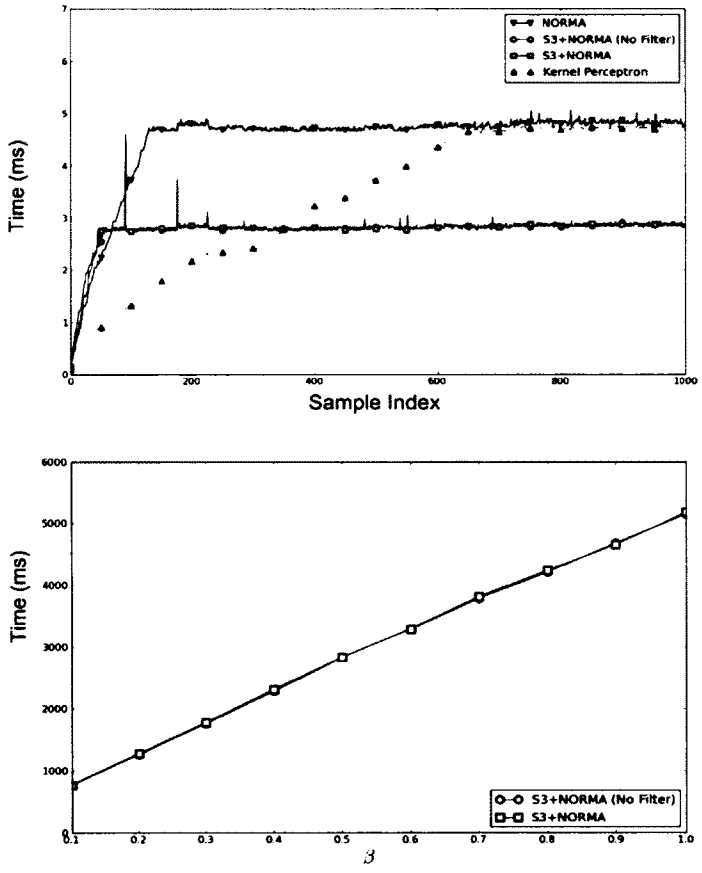


**Figure 18:** In the upper graph time is expressed in milliseconds taken to learn a single sample. Timing is dynamic depending on the training algorithm used and becomes more deterministic when the buffer is eventually filled. S3 parameters chosen are $\beta = 0.5$, $a = 0.0$. In the lower graph time is the number of milliseconds to examine the entire 1000 sample dataset.

**Table 4:** Classification results for 2D stationary and switching data.

| *2D Stationary Dataset* | | | | |
|---|---|---|---|---|
| Training Algorithm | Parameters | Mean Error | Max Error | Min Error |
| NORMA | $\lambda = 0.0$ | 12.3% | 12.3% | 12.3% |
| NORMA+S3 (no output filter) | $\lambda = 0.0$ | 19.52% | 21.4% | 17.7% |
| NORMA+S3 | $\lambda = 0.0$ | 19.4% | 21.2% | 17.2% |
| NORMA | $\lambda = 0.1$ | 20.2% | 20.2% | 20.2% |
| NORMA+S3 (no output filter) | $\lambda = 0.1$ | 25.4% | 26.4% | 23.9% |
| NORMA+S3 | $\lambda = 0.1$ | 27.68% | 29.5% | 25.2% |
| Kernel Perceptron | N/A | **10.5%** | 10.5% | 10.5% |
| *2D Switching Dataset* | | | | |
| Training Algorithm | Parameters | Mean Error | Max Error | Min Error |
| NORMA | $\lambda = 0.0$ | 25.6 | 25.6% | 25.6% |
| NORMA+S3 (no output filter) | $\lambda = 0.0$ | 25.57 | 25.7% | 24.5% |
| NORMA+S3 | $\lambda = 0.0$ | 28.59 | 29.0% | 28.3% |
| NORMA | $\lambda = 0.1$ | 25.6 | 25.6% | 25.6% |
| NORMA+S3 (no output filter) | $\lambda = 0.1$ | **25.47** | 25.7% | 25.3% |
| NORMA+S3 | $\lambda = 0.1$ | 28.54 | 28.8% | 28.3% |
| Kernel Perceptron | N/A | 34.6 | 34.6% | 34.6% |

The use of subset selection produces a classifier that has similar accuracy to the NORMA with $\lambda = 0.1$. This effect can be attributed to the time based penalization that the NORMA uses on the kernel expansion weights to enforce regularization (see Equations (45) and (46)). The S3 algorithm enforces the time based penalization in having a higher probability of selecting more recent support vectors to include into the execution subset. This has the benefit of using fewer support vectors for execution therefore reducing the execution time. The fast drifting data experiment is an extreme case where all classifiers have poor performance when tracking the drift in the data. The benefit of regularization is evident here because the Kernel Perceptron algorithm has no regularization technique and it has the worst error rate.

### 3.7.2 Classification Results of Optical Character Recognition

The MNIST handwritten digit dataset [59] consists of 60,000 training patterns, and 10,000 test patterns. Each pattern is a 28 x 28 grey scale image of a hand written

**Table 5:** Classification results for 2D slow and fast drifting non-stationary data.

| *2D Slow Drifting Dataset* | | | | |
|---|---|---|---|---|
| Training Algorithm | Parameters | Mean Error | Max Error | Min Error |
| NORMA | $\lambda = 0.0$ | 27.6% | 27.6% | 27.6% |
| NORMA+S3 (no output filter) | $\lambda = 0.0$ | **20.79%** | 21.6% | 20.1% |
| NORMA+S3 | $\lambda = 0.0$ | 25.11% | 25.8% | 24.0% |
| NORMA | $\lambda = 0.1$ | 22.1% | 22.1% | 22.1% |
| NORMA+S3 (no output filter) | $\lambda = 0.1$ | 22.16% | 22.3% | 22.0% |
| NORMA+S3 | $\lambda = 0.1$ | 25.72% | 26.0% | 25.5% |
| Kernel Perceptron | N/A | 34.8% | 34.8% | 34.8% |
| *2D Fast Drifting Dataset* | | | | |
| Training Algorithm | Parameters | Mean Error | Max Error | Min Error |
| NORMA | $\lambda = 0.0$ | 48.2% | 48.2% | 48.2% |
| NORMA+S3 (no output filter) | $\lambda = 0.0$ | 46.15% | 47.6% | 44.9% |
| NORMA+S3 | $\lambda = 0.0$ | 48.63% | 49.9% | 47.8% |
| NORMA | $\lambda = 0.1$ | **44.8%** | 44.8% | 44.8% |
| NORMA+S3 (no output filter) | $\lambda = 0.1$ | 45.2% | 45.8% | 44.6% |
| NORMA+S3 | $\lambda = 0.1$ | 45.71% | 46.1% | 45.1% |
| Kernel Perceptron | N/A | 49.3% | 49.3% | 49.3% |

numeral ranging from 0 to 9. In the MNIST dataset, each image has been resized and filtered so that the handwritten numeral is centered within the image. The MNIST dataset is a subset of the National Institute of Standards and Technology (NIST) handwritten digit dataset. Figure 19 illustrates sample images from the training pattern portion of the MNIST dataset. The MNIST dataset is comprised of vectors with 784 dimensions and is used to benchmark machine learning algorithms. For these experiments, a buffer size of 50 was used for all kernel machines. Two hundred samples of either the digit 8 or 1 were randomly drawn from the training set and were used as the sequential stationary dataset for our experiments, e.g. Figure 19 caption (a). A switched dataset was created by reversing the class labels every 50 time steps.

Figure 19, caption (b) illustrates two-fold cross validation, and optimization on the empirical risk to determine the optimal kernel machine parameters. Caption (c) shows how the S3 algorithm does not offer any reduced empirical risk for stationary data. Caption (d) reveals that in the presence of rapid switching, that the S3 algorithm

**Table 6:** Classification results for MNIST data.

| *200 Sample Stationary MNIST Dataset* | | | | |
|---|---|---|---|---|
| Training Algorithm | Parameters | Mean Error | Max Error | Min Error |
| NORMA | $\lambda = 0.0$ | **5.5%** | 5.5% | 5.5% |
| NORMA+S3 (no output filter) | $\lambda = 0.0$ | 9.85% | 11.5% | 8.5% |
| NORMA+S3 | $\lambda = 0.0$ | 10.45% | 12.0% | 8.0% |
| Kernel Perceptron | N/A | 6.0% | 6.0% | 6.0% |
| *200 Sample Switching MNIST Dataset* | | | | |
| Training Algorithm | Parameters | Mean Error | Max Error | Min Error |
| NORMA | $\lambda = 0.0$ | 39.5% | 39.5% | 39.5% |
| NORMA+S3 (no output filter) | $\lambda = 0.0$ | **25.7%** | 28.0% | 23.0% |
| NORMA+S3 | $\lambda = 0.0$ | 28.5% | 31.0% | 26.0% |
| Kernel Perceptron | N/A | 26.5% | 26.5% | 26.5% |

can produce a classifier with lower empirical risk at many different values of $\beta$, and $a$. Since $\beta$ directly influences the computational complexity of the kernel machine we recommend choosing $\beta$ based on design requirements, then select an appropriate stochastic selection rate, $a$, to minimize the empirical risk of the classifier. Figure 19 caption (e), plots the number of milliseconds taken to learn a single sample.

Table 6 summarizes the classifier accuracy on our subset of MNIST samples. The NORMA classifier had the best performance in the stationary case, and when the S3 algorithm was used in the switching scenario, the accuracy of the classifier was greatly enhanced due to the regularization effect. The positive effect of the output filtering in stationary cases, and negative effect in non-stationary cases is repeated with MNIST results as well. It is important to note that the kernel perceptron was the second best classifier in the switching case. The difference between the NORMA ( $\lambda = 0.0$ ) and the kernel perceptron is that the NORMA has a large margin of separation between classes. The results in Table 6 indicate a large margin in counter productive in terms of classification rate when a switch occurs in the data source.

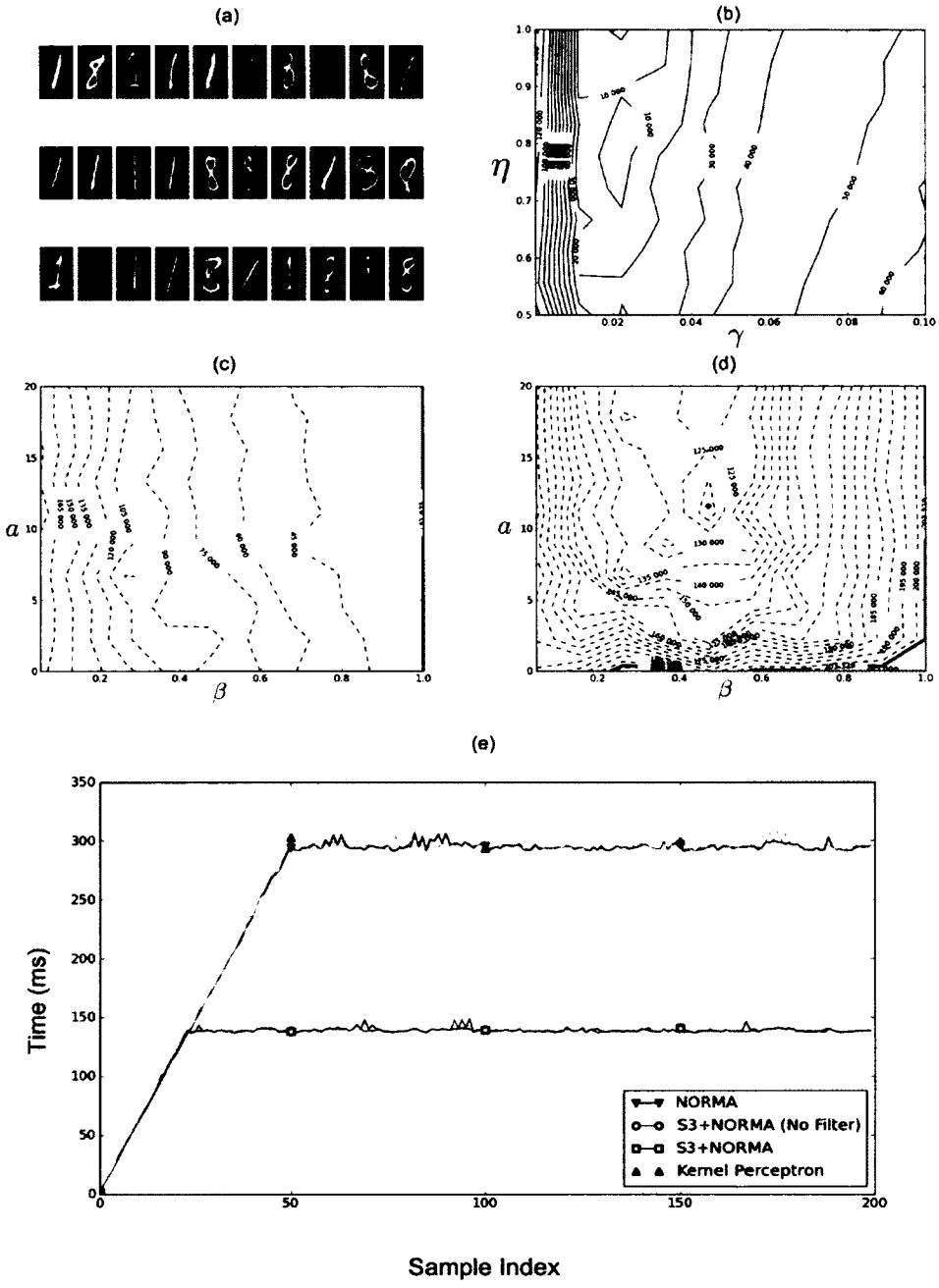


**Figure 19:** (a): Some random samples from the MNIST dataset. (b): Optimum point after doing 2-fold cross validation at $\gamma = 0.02$,$\eta = 1.0$ (c): Stationary set cannot be optimized by the S3 algorithm. (d): S3 operating point at $\beta = 0.47$,$a = 11.6$. (e): Number of milliseconds taken to learn a single sample.

### 3.7.3 Classification Results of the SEA Dataset

The Streaming Ensemble Algorithm (SEA) dataset [60] is commonly used to evaluate machine learning in online, streaming environments. The SEA dataset consists of 60,000 three dimensional vectors having values between 0.0 and 10.0, with four different concepts. The dataset is broken into four segments of 15,000 points each, and are labeled as +1 according to $\mathbf{x}(1) + \mathbf{x}(2) \leq \theta$. For each of the blocks, values for $\theta$ are 8, 9, 7, and 9.5 [60].

Cross validation was inappropriate in this case due to the drifting nature of the data. The first 2000 samples from the SEA dataset were taken and the kernel machine parameters were optimized to minimize the empirical risk function. Figure 20, caption (a) illustrates the optimal parameters that exhibit the lowest empirical risk with respect to the first 2000 samples of the dataset.

Figure 20, caption (b) shows how the parameter selection for the S3 algorithm affects the empirical risk of the classifier. There is no parameterization that will yield a classifier with lower empirical risk than the classifier using it's entire buffer. The contour lines are warped nearer the upper right corner of the figure. A parameterization was chosen that is close to the upper right corner of the plot. A buffer of 100 samples was used for all experiments with the SEA dataset.

Table 7 gives the classification results involving the SEA dataset. The classifier using the S3 algorithm exhibits only a marginal improvement to its classification rate. The S3 algorithm can perform better than a classifier with a fixed buffer when there is a concept change present in the data. But since the S3 classifier uses less kernel terms in the evaluation of the kernel expansion, the stochastic noise and lack of expressiveness in the classification output will generally cause a kernel machine using the S3 algorithm to perform worse than the kernel machine operating by itself.

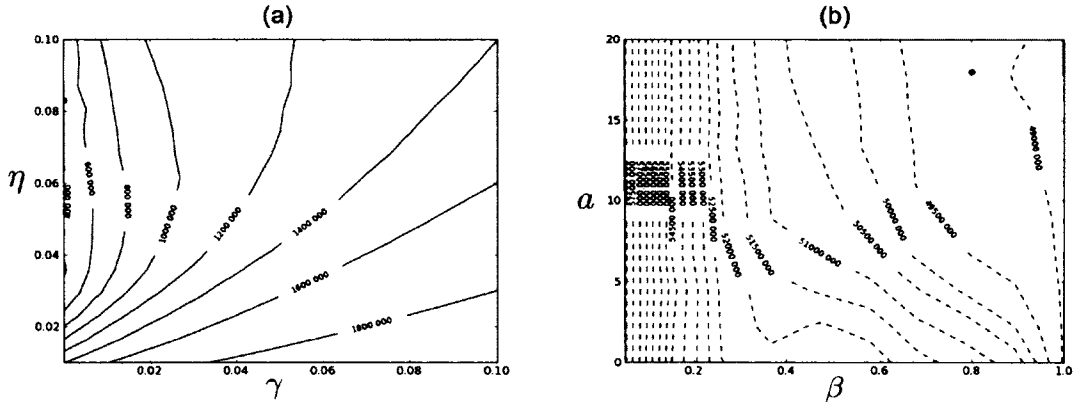


**Figure 20:** (a):Optimum found at $\gamma = 6.336e-05$, $\eta = 0.083$. (b): S3 operating point at $\beta = 0.8$,$a = 18.0$.

**Table 7:** Classification results for SEA dataset.

| *60,000 Sample SEA Dataset* | | | | |
|---|---|---|---|---|
| Training Algorithm | Parameters | Mean Error | Max Error | Min Error |
| NORMA | $\lambda = 0.0$ | 37.6% | 37.6% | 37.6% |
| NORMA+S3 (no output filter) | $\lambda = 0.0$ | 37.57% | 37.61% | 37.50% |
| NORMA+S3 | $\lambda = 0.0$ | 38.37% | 38.39% | 38.32% |
| Kernel Perceptron | N/A | **28.58%** | 28.58% | 28.58% |

## 3.8 Conclusions and Applications

Based on the results given in Tables 4, 5, 6, 7 the S3 algorithm is useful in settings where concept drift is present. Lower computational requirements can be achieved with the use of stochastic indexing in the support vector buffer.

The two primary drawbacks of the S3 algorithm are the parameters $a$ and $\beta$ are chosen by offline optimization and the S3 algorithm has a larger misclassification rate for stationary data sources.

The S3 algorithm allows for an efficient method of selecting a subset of support vectors for evaluation of a kernel machine classifier. The resulting classifier produced by the S3 algorithm has additive noise that is inversely proportional to the size of the subset used for evaluation of the kernel classifier. The use of exponentially distributed random variates allows for the S3 algorithm to focus on more recently acquired SVs, allowing the classifier to track *switching* and *drifting* changes effectively and efficiently. Using the principle of recent data being more relevant than older data is an effective approach for online kernel machines.

The S3 algorithm can be used in conjunction with any online kernel machine training technique. It is important for online kernel machines to be computationally efficient because of the real time requirements of online environments. The S3 algorithm is an important contribution because it scales linearly with the number of training samples, and is compatible with other time based regularization approaches.

The S3 algorithm is a novel contribution in that it differs from the ensemble approaches used in the works of [60, 65–68] by using a single kernel machine for learning and evaluation. The S3 algorithm also distinguishes itself from the work of Klinkenberg and Joachims [57] by addressing the problem of computational complexity and adapting to concept drift simultaneously.

While the S3 algorithm still operates in $O(m)$ computational complexity, there

are many instances where a scalar improvement is beneficial. Computer vision is an example where a doubling or tripling of processing frame rate is beneficial. A system designer has to make the decision on the trade-off between computational complexity and empirical risk.

The application area for the S3 algorithm is hybrid systems that require pattern recognition in an online environment. Online systems such as computer vision, robotics, telecommunications, mobile computing, and embedded control systems are examples of such systems. It is necessary for the S3 algorithm to be combined with concept drift detection [72] and an online method for adapting the S3 parameters is also needed.

# Chapter 4

# The Partitioned Kernel Machine Algorithm

The previous chapter investigated the use of a subset technique for a binary classification system using kernel machines in an online environment. This chapter makes use of a subset selection method that is based on a similarity measure with the current kernel machine input. The purpose of this chapter is to develop a training algorithm for an online kernel machine that will provide greater accuracy than current techniques while still being adaptable to the input, $\mathbf{x}_t$.

## 4.1 Chapter Outline

The use of partitioning within the kernel machine buffer is examined in this chapter. The partitioning is done during each time step and it involves a similarity measure with the current input to the kernel machine. The first section introduces the partitioned kernel machine algorithm. The second section gives experimental results of the partitioned kernel machine algorithm.

## 4.2 Partitioned Kernel Machine Algorithm

The fundamental component of the Partitioned Kernel Machine (PKM) algorithm is an update procedure that focuses on a subset of the stored vectors in the kernel machine buffer. This is in contrast to standard online kernel machine algorithms because the NORMA and the KLMS algorithm only update the weight of the current input vector at time $t$. In the PKM algorithm the subset of $\alpha$'s associated with the vectors are updated in place within the kernel machine buffer in two stages. First, a similarity function is defined between two vectors as $r : \Re^a \times \Re^a \rightarrow \Re$. The similarity measure is the Gaussian kernel since all kernel pairs are already computed with each kernel machine evaluation. Once all vectors in the kernel machine buffer are assigned a similarity score to the current input, a percentage of the top scored vectors have their $\alpha$ weights repeatedly updated. This process ensures that common vectors in the kernel machine buffer are updated more frequently than outliers.

The KLMS algorithm presented in the work of Liu et al. [53] does not describe a method for updating kernel machine weights in place.

Many parallels can be drawn between the kernel least mean square (KLMS) algorithm [53] presented in the work of Liu et al., and the naive online $R_{reg}$ algorithm [40] (NORMA) presented in the work of Kivinen et al. This derivation follows the work of [40] while using the work of [53] to make claims of stability and self regularization.

To begin the derivation of this algorithm the definition of empirical risk commonly defined in statistical learning theory [1], [3] is used. Equation (58) defines the empirical risk of a kernel machine. The empirical risk defines the average amount of loss incurred for the kernel machine. The loss function used in this work is the squared loss, and is defined in Equation (59).

$$R_{emp}[f_t] = \frac{1}{m} \sum_{i=1}^{m} l(f_t, \mathbf{x}_i, y_i) \tag{58}$$

$$l(f_t, \mathbf{x}, y) = \frac{1}{2}(y - f_t(\mathbf{x}))^2 \tag{59}$$

In a similar motivation to the work of Kivinen et al. [40], stochastic gradient descent is applied to the current prediction function, minimizing the empirical risk. The problem with the expression for $R_{emp}$, is that it requires evaluation of $f_t$ on every pattern in the kernel machine buffer. The computational complexity scales in $O(m^2)$ to perform a single iteration of the training algorithm. Equation (60) defines another risk functional that is similar to the "instantaneous" risk defined in [40]. The key difference is that this particular risk functional defines the loss incurred for any expansion term in the buffer.

$$\hat{R}_i[f_t] = l(f_t, \mathbf{x}_i, y_i) = \frac{1}{2}(y_i - f_t(\mathbf{x}_i))^2 \tag{60}$$

Equations (61), (62), (63), and (64) give the derivations of the update equations for the $i$th kernel expansion term. Equation (61) begins the update procedure with an update to the kernel machine prediction function. The update is performing stochastic gradient descent on $\hat{R}_i[f_t]$, therefore any expansion term in the buffer can be chosen for optimization. If one were to optimize every parameter in the buffer, this can be considered a full iteration of gradient descent. Optimizing the kernel machine across all terms in the buffer with $n$ iterations would have $O(nm^2)$ computational complexity. This algorithm will select a subset of size $j$ terms yielding a computational complexity of $O(njm)$.

$$f_t' \leftarrow f_t - \eta \frac{\partial\left(\hat{R}_i[f_t]\right)}{\partial f_t} \tag{61}$$

Equation (61) denotes the updated prediction function to be $f_t'$. Since an existing expansion term is chosen, there is no new support vector being added to the buffer.

The time index $t$ stays the same, and the expansion term weight $\boldsymbol{\alpha}(i)$ will be updated.

$$f_t^{'} \leftarrow f_t + \eta\left(y_i - f_t\right) \tag{62}$$

Equation (62) reveals that the update to the kernel machine will be scaled error between the stored value, $y(i)$ and the prediction of $y(i)$) at time t. The use of gradient descent for training kernel machines was explored in [1] and [40]. The key difference between this work and that of [1] and [40] is that there is no form of explicit regularization of the risk functional in this case.

$$\sum_{b=1}^{m} \boldsymbol{\alpha}(b) k\left(\mathbf{x}_i, \mathbf{x}_b\right) = \sum_{b=1}^{m} \boldsymbol{\alpha}(b) k\left(\mathbf{x}_i, \mathbf{x}_b\right) + \eta\left(y_i - f_t\left(\mathbf{x}_i\right)\right) \tag{63}$$

Equation (63) re-writes Equation (62) in terms of their weight parameters by fixing all other indices of the expansion terms. In this case $m$ refers to the size of the buffer used, and the first term in the buffer is indexed at 1.

$$\boldsymbol{\alpha}^{'}(i) \leftarrow \boldsymbol{\alpha}(i) + \eta \frac{\left(y_i - f_t\left(\mathbf{x}_i\right)\right)}{k(\mathbf{x}_i, \mathbf{x}_i)} \tag{64}$$

Throughout this chapter a radial basis function kernel is used, where $k(\mathbf{x}_i, \mathbf{x}_i) = 1.0$. The final update is expressed in Equation (65).

$$\boldsymbol{\alpha}^{'}(i) \leftarrow \boldsymbol{\alpha}(i) + \eta\left(y_i - f_t\left(\mathbf{x}_i\right)\right) \tag{65}$$

The update rule is of the exact same form as the update rule for the kernel adatron algorithm [47]. The kernel adatron algorithm is a batch algorithm for the training of kernel machine classifiers, while the KLMS algorithm is specifically designed for dealing with online use for regression applications. An update rule is needed for new data produced from the time embedding process, $f_{t+1} \leftarrow f_t$. The instantaneous risk at time $t$ is used to update the prediction function. Equation (63) can be directly used

for updating the kernel machine prediction function to the next time step. Equation (66) describes the update $f_{t+1} \leftarrow f_t$, which is identical to that of KLMS [53] [51].

$$\boldsymbol{\alpha}(t) \leftarrow \eta(y_i - f_t(\mathbf{x}_t)) \tag{66}$$

The partitioned kernel machine (PKM) algorithm creates a partition of the most similar vectors to the current input for updating with Equation (65). A time based stochastic subset technique for kernel machine classifiers was discussed in Chapter 3. The PKM algorithm differs significantly from the subset technique in Chapter 3 in that it uses a similarity measure with the input ($\mathbf{x}_t$) to select support vector weights for update. The PKM algorithm must be used following the KLMS algorithm, therefore the PKM algorithm is an enhancement to the traditional KLMS algorithm. The PKM algorithm is described in Algorithm 3.

The PKM algorithm also has a common characteristic to the kernel affine projection algorithm (type I) (KAPPA-1) [51]. In KAPPA-1, the operation of KLMS is improved by updating a block of $j$-expansion terms from $\mathbf{x}_{t-j+1}$ to $\mathbf{x}_t$, in addition to adding the current term, $\mathbf{x}_t$.

Based on the description of Algorithm 3, the computational complexity of the PKM algorithm is in the order of $O(m \log m)$. If the computational complexity of the kernel evaluations outweigh the sorting operation in PKM, then the overall computational complexity will be very similar to $O(m)$. The execution time will be explored in our experimental results.

## 4.3 Experiments

The PKM algorithm was tested on two different datasets. The first dataset involves non-linear function approximation with noise. The second dataset involves the prediction of a benchmark chaotic time series. Different datasets were used in this chapter

**Algorithm 3** The PKM Algorithm.

Input:

- $p$: percentage of top similarity scores to partition.
- $i$: Number of updates performed across the partitioned vectors.
- $X$: Buffer of vectors used in the KLMS algorithm.
- $A$: Buffer of training weights used in the KLMS algorithm.
- $\mathbf{x}_n$: Current input for the kernel machine.

Initialize:

- Ordered Set, $Keys = \emptyset$

**for** $j = 1 \rightarrow m - 1$ **do**
  $\mathbf{x}_j \leftarrow X(j)$
  $Keys \leftarrow \{r(\mathbf{x}_j, \mathbf{x}_n), j\}$
**end for**
Sort $Keys$ in descending order based on similarity value in each tuple.
**for** $k = 1 \rightarrow i$ **do**
  **for** $j = 1 \rightarrow \lfloor (m-1)p \rfloor$ **do**
    $\{r(\mathbf{x}_{index}, \mathbf{x}_n), index\} \leftarrow Keys(j)$
    Update $A(index)$ by Equation (65).
  **end for**
**end for**

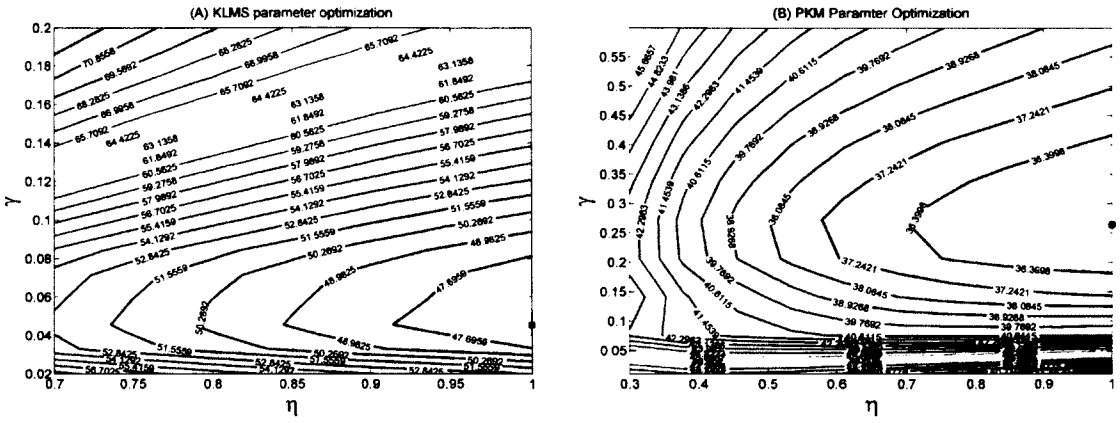


**Figure 21:** Panel A: The optimal parameters for the KLMS algorithm were found at $\eta = 1.0$, $\gamma = 0.04536$. Panel B: The optimal parameters for the PKM algorithm ($p = 0.01$, $i = 3$) were found at $\eta = 1.0$, $\gamma = 0.2640$. All parameters were found by performing a two dimensional pattern search across $\eta$ and $\gamma$. The pattern search minimized the total absolute error (shown above by contour lines) of the kernel machine. The size of $\eta$ is limited to 1.0 to maintain numerical stability [53]. All parameterizations and results are summarized in Table 10 in Section 4.4.

when compared to the previous chapter because KLMS and PKM are examples of kernel machines used for regression, not classification.

The parameters $\eta$ and $\gamma$ are optimized to give the lowest possible total absolute error before investigating the effect of the PKM parameterizations. Two examples of the optimization of $\eta$ and $\gamma$ are shown in Figure 21. The optimization of $\eta$ was confined to $< 1.0$ for stability requirements [51]. Finally a timing analysis was performed to investigate the computational properties of the PKM algorithm.

### 4.3.1 Function Learning

Equation (67) describes the continuous function that will generate the discrete time series. Noise was added to the time series with a mean of 0.0 and standard deviation of 0.01. The function was sampled at 50Hz and ran from 0.0 to 10.0 seconds, producing a time series with 501 points. A time embedding dimension of $a = 10$ was used.

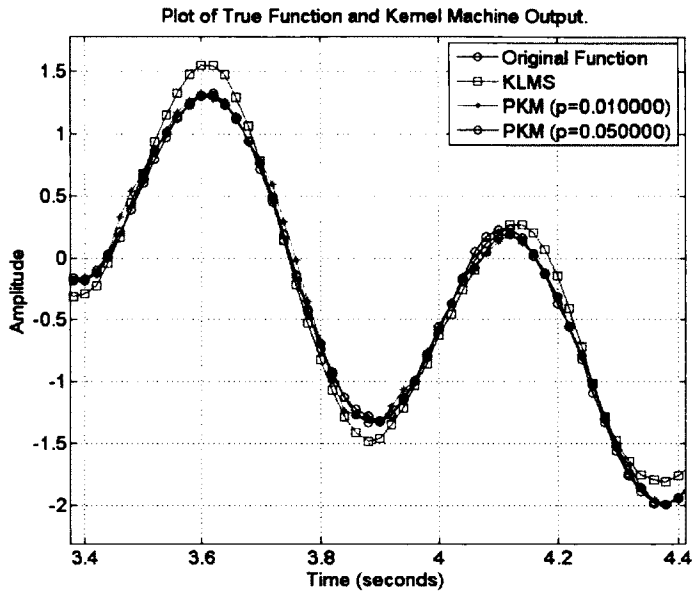


**Figure 22:** The following kernel machines were used to approximate the time series for the continuous function $ts$: KLMS ($\eta = 1.0$, $\gamma = 0.04536$), PKM ($\eta = 1.0$, $\gamma = 0.2640$, $p = 0.01$, $i = 3$), PKM ($\eta = 1.0$, $\gamma = 0.1692$, $p = 0.05$, $i = 3$). The PKM with $p = 0.01$ and $p = 0.05$ update 1% and 5% of the most similar vectors stored in the kernel machine respectively.

$$ts(t) = \sin(4\pi t) + \sin(0.8\pi t) + \mathcal{N}(0.0, 0.01) \tag{67}$$

Figure 22 illustrates the original time series from 0.9 seconds to 2.5 seconds. The following kernel machines were tested: KLMS ($\eta = 1.0$, $\gamma = 0.04536$), PKM ($\eta = 1.0$, $\gamma = 0.2640$, $p = 0.01$, $i = 3$), PKM ($\eta = 1.0$, $\gamma = 0.1692$, $p = 0.05$, $i = 3$). All three kernel machines exhibit errors in approximating the time series initially but the errors diminish as time progresses.

The error curves for the three separate kernel machines are shown in Figure 23. The error curves show two interesting features. First, it can be seen that the PKM errors drop dramatically at about 2.75 seconds, indicating the speed at which the PKM has adapted to estimating the function. Second, the PKM using $p = 5\%$

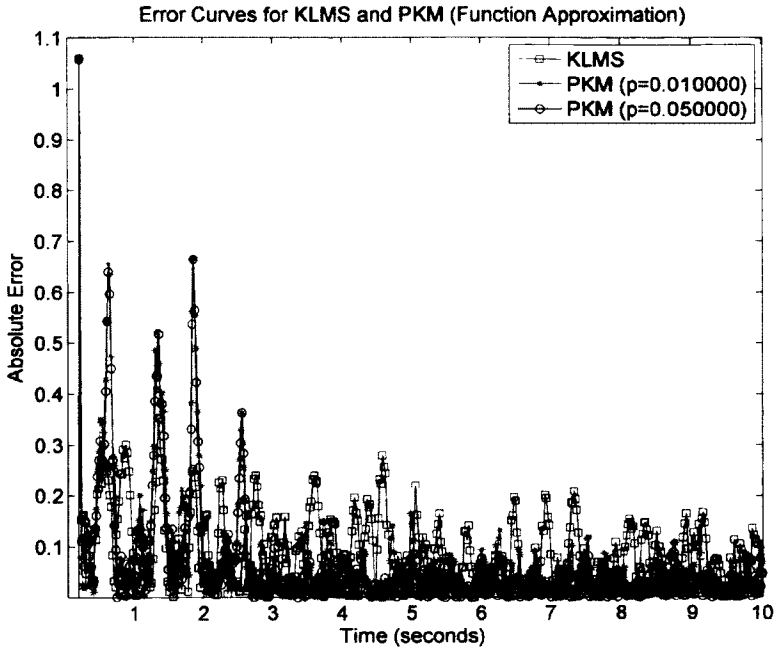


**Figure 23:** A plot of errors incurred by all three kernel machines. The PKM algorithm has an advantage over the KLMS algorithm used alone. As $p$ is increased, the overall accuracy is improved.

outperforms $p = 1\%$, indicating that updating more of the stored vectors in the buffer is correlated with a better error performance. The improvement to error performance comes at a computational cost. The advantage of using a similarity measure to partition the vectors stored in the kernel machine is investigated in Section 4.4.

Table 8 shows that updating more of the stored vector weights in place yields less total error in the function approximation.

**Table 8:** Summary of Function Learning Experimental Results

| Experiment | Kernel Machine Algorithm | Total Error |
|---|---|---|
| Function Learning | KLMS ($\eta = 1.0$, $\gamma = 0.04536$) | 46.1617 |
| Function Learning | PKM ($\eta = 1.0$, $\gamma = 0.2640$, $p = 0.01$, $i = 3$) | 35.3178 |
| Function Learning | PKM ($\eta = 1.0$, $\gamma = 0.1692$, $p = 0.05$, $i = 3$) | **27.8368** |

**Table 9:** Summary of MGTS Experimental Results

| Experiment | Kernel Machine Algorithm | Cumulative Error |
|---|---|---|
| Chaotic Time Series | KLMS ($\eta = 0.7400$, $\gamma = 0.7467$) | 25.1368 |
| Chaotic Time Series | PKM ($\eta = 1.0$, $\gamma = 0.5103$, $p = 0.01$, $i = 3$) | 15.3680 |
| Chaotic Time Series | PKM ($\eta = 1.0$, $\gamma = 0.4259$, $p = 0.05$) | **11.2531** |

### 4.3.2 Chaotic Time Series Prediction

The Mackey-Glass chaotic time series is often used as a benchmark test in time series forecasting applications and is described by the differential equation given in Equation (68). Simulink was used to generate the time series with the following parameters: $b = -0.1$, $c = 0.2$, $\tau = 30$, $ts(0.0) = 1.2$, start time = 0.0, stop time = 6000, fixed time step = 6.0, and solver = ode3. To each sample a small amount of normally distributed random noise was added with $\mu = 0.0$ and $\sigma^2 = 0.01$. A time series of 1001 points was generated, and a time embedding dimension of $a = 15$ was used.

$$\frac{d\left(ts\left(t\right)\right)}{dt} = -bts\left(t\right) + \frac{cts\left(t-\tau\right)}{1+ts\left(t-\tau\right)^{10}} \tag{68}$$

Parameter optimization and experiments involving accuracy were performed on the time range of $t = 1200$ to $t = 3600$ (time series points 200 to 600). Figure 24 shows a plot of the Mackey-Glass time series. The results are similar to the previous section in that the PKM outperforms the KLMS algorithm in terms overall accuracy. The use of a similarity measure to partition the vectors stored in the kernel machine is investigated in Section 4.4.

Table 9 gives the cumulative error of the kernel machines when approximating the Mackey-Glass time series. As with the previous experiment, updating more expansion weight in place yields a lower cumulative error.

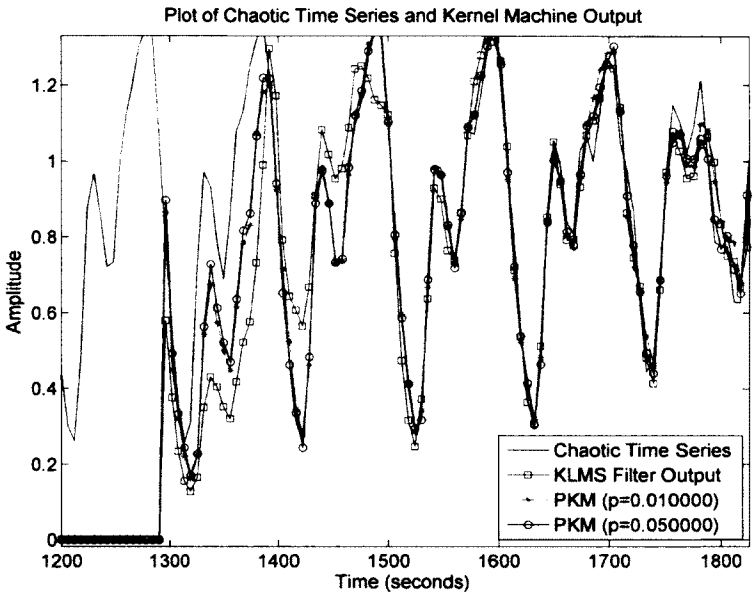


**Figure 24:** A time plot of the Mackey-Glass time series and the time series estimates from both the KLMS and PKM algorithms. The PKM parameters for this experiment were $i = 3$ and $p = 0.05$.

## 4.4 Summary of Results

Table 10 shows the cumulative errors of both the function approximation and the chaotic time series experiments. The total absolute error was reduced by approximately 30%-50% by using the PKM algorithm. It is worth noting that while the use of a larger value of $p$ will yield better performance in terms of overall error, the extra computational cost in updating more vector weights may be a significant design decision.

As well Table 10 illustrates the improvement of using a similarity score when partitioning the kernel machine while training. The 5th and and 11th rows from the top in Table 10 illustrate the advantage of using a similarity score to partition the kernel machine verses random partitioning or KAPPA-1. A better error performance can be achieved when using the similarity score as a heuristic for selection of vectors to be updated. PKM performs better with $i = 5$ and $p = 0.01$ verses random partitioning

with twice the number of iterations ($i = 10$ vs $i = 5$) and five times the partition size ($p = 0.05$ vs $p = 0.01$). KAPPA-1 results in the worst error performance for the same parameterization.

A possible explanation of KAPPA-1's poor performance is that both experiments deal with periodic type functions. Updating a block of more recent vectors will have the least advantage when updating the coefficient weights because similar inputs will not be observed until further in the future.

**Table 10:** Summary of Experimental Results

| Experiment | Kernel Machine Algorithm | Total Error |
|---|---|---|
| Function Learning | Random Partitioning ($\eta = 1.0$, $\gamma = 0.2640$, $p = 0.01$, $i = 5$) | 35.1685 |
| Function Learning | PKM ($\eta = 1.0$, $\gamma = 0.2640$, $p = 0.01$, $i = 5$) | 31.0594 |
| Function Learning | KAPPA-1 ($\eta = 1.0$, $\gamma = 0.2640$, $p = 0.01$, $i = 5$) | 68.6555 |
| Function Learning | Random Partitioning ($\eta = 1.0$, $\gamma = 0.1692$, $p = 0.05$, $i = 10$) | 31.7252 |
| Function Learning | PKM ($\eta = 1.0$, $\gamma = 0.1692$, $p = 0.05$, $i = 10$) | **21.0389** |
| Function Learning | KAPPA-1 ($\eta = 1.0$, $\gamma = 0.1692$, $p = 0.05$, $i = 10$) | 172.8519 |
| Chaotic Time Series | Random Partitioning ($\eta = 1.0$, $\gamma = 0.5103$, $p = 0.01$, $i = 5$) | 19.5716 |
| Chaotic Time Series | PKM ($\eta = 1.0$, $\gamma = 0.5103$, $p = 0.01$, $i = 5$) | 14.1370 |
| Chaotic Time Series | KAPPA-1 ($\eta = 1.0$, $\gamma = 0.5103$, $p = 0.01$, $i = 5$) | 26.6758 |
| Chaotic Time Series | Random Partitioning ($\eta = 1.0$, $\gamma = 0.4259$, $p = 0.05$, $i = 10$) | 14.7369 |
| Chaotic Time Series | PKM ($\eta = 1.0$, $\gamma = 0.4259$, $p = 0.05$, $i = 10$) | **9.2517** |
| Chaotic Time Series | KAPPA-1 ($\eta = 1.0$, $\gamma = 0.4259$, $p = 0.05$, $i = 10$) | 64.1635 |

Figure 25 illustrates the growth of the computational complexity as more vectors are added to the kernel machine buffer. Timing measurements were taken with each input vector while running the function learning experiment described in Section 4.3.1. It is evident that the execution time scales in the order of $O(m)$ for all three kernel machines. The computational effect of sorting the similarity values in the PKM algorithm is negligible when compared to kernel function evaluation, when considering a small enough time series (small $m$).

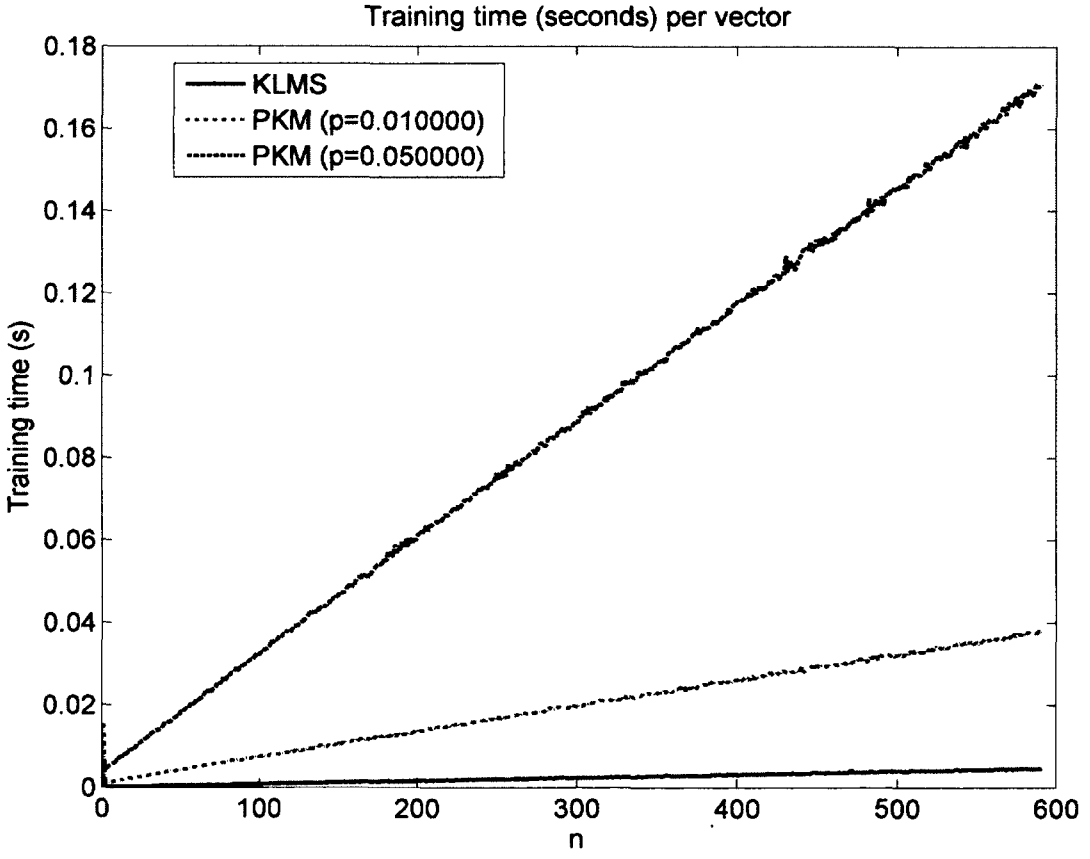


**Figure 25:** The execution time scales in the order of $O(m)$ for all three kernel machines. The computational effect of sorting the similarity values in the PKM algorithm is negligible when compared to kernel function evaluation when considering a small enough time series.

## 4.5 Conclusions and Applications

The results from the experiments indicate that selecting a subset of support vectors for additional optimization yields an increase in accuracy at the cost of increased computational effort. The PKM algorithm scales in $O(t)$ computational complexity as long as the size of the partition is held constant, making it suitable for online applications. Table 10 shows the benefit the similarity measure for choosing the partition. In all cases the similarity measure produced a more accurate kernel machine than the use of randomized partitioning when kernel machine parameters were held constant.

The PKM algorithm is intended to be used together with the KLMS algorithm to improve the overall accuracy in kernel machine regression applied to time series data. The use of a similarity measure between vectors stored in the kernel machine buffer and the current input vector allows for more common vectors to be updated more often. Kernel machines represent an important area of machine learning research, and have many useful applications including time series prediction. The results achieved in this chapter illustrate that the proposed PKM algorithm can achieve superior error performance while still scaling linearly in computational effort.

The PKM algorithm is a significant novel contribution because it extends upon the use of stochastic gradient descent in online environments. In the work of Liu et al. [51, 53], KLMS is shown to be an effective algorithm for non-linear time-series forecasting. The PKM algorithm advances the use of KLMS in two ways. The first way is by providing in-place updates of expansion term weights which allows the error to be reduced significantly. The second contribution is the use of a similarity measure with the current input which further increases the accuracy of the kernel machine.

The PKM algorithm algorithm was designed to be used in conjunction with the KLMS algorithm for online regression tasks. The application scope for kernel machine

regression is diverse since kernel machines are well suited for estimating non-linear relationships. Problems that require online non-linear regression are non-linear filtering, computer vision, non-linear embedded control systems, telecommunications, and signal processing to name a few.

# Chapter 5

# Smooth Delta Corrected Kernel Machine

In online applications involving kernel machines there are usually two goals that a system designer is trying to meet. First, the system needs to be computationally efficient. In other words, the system must be able to keep up with the stream of incoming data. Second, there will be a finite amount of memory associated with the system. It is the goal of this chapter to investigate the effect of memory buffer truncation error on the performance of the KLMS algorithm in order to correct for truncation error.

## 5.1 Chapter Outline

Preliminary material on the Kernel Least Mean Square (KLMS) [53] [51] algorithm and on the Naive Online $R_{reg}$ Minimization Algorithm [40] (NORMA) can be found in Section 2.8 and 2.6 respectively. The KLMS algorithm offers an computationally efficient method for training kernel machines for time series prediction. For preliminary material on the concept of time-embedding in the use of kernel machines for regression and time series prediction applications please refer to Section 2.7.

Section 5.2 will explore the topic of truncation error in terms of a finite memory buffer. After exploring the problem of truncation error, Section 5.3 will introduce the

Smoothed Delta Compensated Kernel Least Mean Square (SDC-KLMS) algorithm, which is a solution to the problem of truncation error with the KLMS algorithm. Section 5.5 of this chapter will give experimental validation of the SDC-KLMS algorithm and concluding remarks will be given in Section 5.6.

## 5.2 The KLMS Algorithm and Truncation Error

The primary goal of an online kernel machine for regression tasks is to make predictions of times series values based on previous observations. The kernel machine estimates a prediction function, $f_t$ and makes a prediction on $y_t$ given the input vector $\mathbf{x}_t$. Equation (69) defines the prediction function as a linear combination of $m$ weighted kernel functions. This linear combination of often referred to as the kernel expansion and each term in the expansion is referred to as an expansion term. The values for $\mathbf{x}_i$ and $\boldsymbol{\alpha}(i)$ are stored in memory for each subsequent prediction, so the execution time of the kernel machine scales linearly in $O(m)$.

$$f_t(\mathbf{x}_t) = \sum_{i=1}^{m} \boldsymbol{\alpha}(i)\, k(\mathbf{x}_t, \mathbf{x}_i) \tag{69}$$

Throughout this thesis it is assumed that the kernel expansion contains $m$ terms and there is a buffer with space for $m$ expansion terms. At each time step the time embedding process will produce a $\mathbf{x}_t$ and $y_t$. Each time step will add $\mathbf{x}_t$ which will update the kernel machine ($f_t \rightarrow f_{t+1}$). Equation (69) can then be re-written as Equation (70), assuming a circular buffer is used where index values are mapped back between 1 and $m$.

$$f_t(\mathbf{x}_t) = \sum_{i=t-m}^{t} \boldsymbol{\alpha}(i)\, k(\mathbf{x}_t, \mathbf{x}_i) \tag{70}$$

In the field of statistical learning, the concept of loss and risk functions are used

extensively. An interested reader can consult [1], [3], and [40] for further reading. In the work of Kivinen et al. [40], the authors propose a bound on the truncation error created by the use of a finite size buffer for storing expansion terms. Given $\|k(\mathbf{x}, \bullet)\| \leq X$ and $\{\exists i : \|\boldsymbol{\alpha}(i)\| \leq C\}$, Equation (71) from [40] places a upper bound on the error between an ideal kernel machine with an infinite buffer, and a kernel machine with a finite buffer.

$$\|f - f_t\| \leq \sum_{i=t-m}^{t} CX = mCX \tag{71}$$

After considering the maximum bound that Equation(71) places on the truncation error, a prediction error can be assigned to the kernel machine output. Assuming that the buffer is full and a new term is to be added by replacing the $i$th term in the expansion, Equation (72) describes the error incurred.

$$\delta = \|f_{t+1} - f_t\| = \boldsymbol{\alpha}(i)k(\mathbf{x}_{t+1}, \mathbf{x}_i) \tag{72}$$

The significance in Equation (72) describes an incurred error in the output of the kernel machine as a result of losing an expansion term due to finite size buffer. The error will be compensated for in the novel kernel machine learning algorithm.

## 5.3 Smoothed Delta Compensated-KLMS (SDC-KLMS) Algorithm

The pseudo-code presented in Algorithm 4 describes the SDC-KLMS algorithm. The SDC-KLMS algorithm combines equations 66, 65, and 72 into a sequential algorithm that compensates for truncation error, while providing faster convergence through the smoothing of the kernel weights. The SDC-KLMS algorithm was designed with

run time efficiency in mind. A circular buffer is used so that any modifications to expansion weights and vectors can be performed in place in computer memory, avoiding costly shifting and copying of data.

SDC-KLMS randomly picks $j$ index values across the entire buffer. Since the truncation error occurs because of a loss of the oldest expansion term, it is better to smoothly distribute expansion term updates across the entire buffer. The computational complexity of adding and new expansion term to the buffer using SDC-KLMS is $O(jm)$ where $j$ is the number of expansion terms to update before adding $\mathbf{x}_t$ and $y_t$.

## 5.4 Stability of KLMS

It is interesting to point out that the KLMS algorithm presented in [53] and [51] is equivalent to the NORMA parameterized with $\lambda = 0.0$, and update rules based on a quadratic loss function. When looking at the stability of SDC-KLMS any assumptions made on the upper bound on $\eta$ still hold as derived in [51]. The need for explicit regularization to prevent over fitting in SDC-KLMS is avoided due to the self-regularizing property of KLMS [51, 53]. The conservative upper bound for $\eta$ in KLMS using an RBF kernel, an infinite buffer, and a total of $N$ observed values, is given in Equation (73) [51].

$$\eta < \frac{N}{\sum_{i=1}^{N} k(\mathbf{x}_i, \mathbf{x}_i)} < 1.0 \tag{73}$$

This conservative upper limit does not take truncation error into account and it is shown experimentally that $\eta < 1.0$ does not guarantee convergence of the KLMS algorithm.

**Algorithm 4** The SDC-KLMS algorithm.

Initialize:

- Circular buffer, $D = \emptyset$, size $m$
- $f_0 = 0$
- $j \leftarrow$ Number of existing expansion terms to optimize.
- $\gamma \leftarrow$ kernel parameter.
- $\eta \leftarrow$ learning rate.

**while** $t < \infty$ **do**
  **if** $t \geq m$ **then**
    At time $t$, take sample and label pair $p \rightarrow (\mathbf{x}_t, y_t)$ from input.
    $z \leftarrow (t+1) \, mod \, (m+1) + 1$
    $r \leftarrow (t) \, mod \, (m+1) + 1$
    **for** $w = 1$ to $j$ **do**
      $q \leftarrow$Uniform random index value, not including $z$.
      $\delta_{z,q} = \boldsymbol{\alpha}(z) k(\mathbf{x}_z, \mathbf{x}_q)$
      $\boldsymbol{\alpha}'(q) \leftarrow \boldsymbol{\alpha}(q) + \eta (y_q - f_t(\mathbf{x}_q + \delta_{z,q}))$
    **end for**
    $\delta_{z,r} = \boldsymbol{\alpha}(z) k(\mathbf{x}_z, \mathbf{x}_r)$
    $\boldsymbol{\alpha}(r) \leftarrow \eta (y_t - f_t(\mathbf{x}_t + \delta_{z,r}))$
  **else**
    At time $t$, take sample and label pair $p \rightarrow (\mathbf{x}_t, y_t)$ from input.
    $z \leftarrow (t+1) \, mod \, (m) + 1$
    $r \leftarrow (t) \, mod \, (m) + 1$
    **for** $w = 1$ to $j$ **do**
      $q \leftarrow$Uniform random index value, not including $z$.
      $\boldsymbol{\alpha}'(q) \leftarrow \boldsymbol{\alpha}(q) + \eta (y_q - f_t(\mathbf{x}_q))$
    **end for**
    $\boldsymbol{\alpha}(r) \leftarrow \eta (y_t - f_t(\mathbf{x}_t))$
  **end if**
  Store at position $r$ in $D \leftarrow y_t$,$\mathbf{x}_t$
**end while**

## 5.5 Experiments

The KLMS algorithm and the novel SDC-KLMS algorithm are tested in three separate settings. In the first setting a time-series representing a sine function in both noisy and ideal conditions is simulated. The second setting is a noisy chaotic time-series that has pseudo-periodic properties. Finally, the third setting is a time series that was acquired from a 6 degree-of-freedom (DOF) haptic instrument [74].

### 5.5.1 Sine Wave Simulations

The first test of the algorithm seeks to show an extreme effect of truncation error. Equation (72), states that the most extreme error will occur when the kernel machine needs to replace a kernel expansion term that has both a large weight term ($\boldsymbol{\alpha}(i)$), and is similar to the current input vector ($\mathbf{x}_t \approx \mathbf{x}_i$). For the first test, $p(t)$ is produced by sampling the function $s(t) = \sin(2\pi t)$ at a sampling rate of 100Hz over a duration of 2.0 seconds. Unless otherwise stated, the time embedding process uses a dimension of 11, $\mathbf{x}_t \leftarrow [p(t-9)\cdots p(t)]$ and $y_t = p(t+1)$. Figure 26 shows the predictions produced by both KLMS and SDC-KLMS. The randomized updates of expansion term weights is a stochastic process, but it represents extra stochastic gradient descent steps that increases the accuracy of the PKM kernel machine compared to traditional KLMS and will not result in reduced accuracy.

The truncation error is evident at $t = 111$. At this point KLMS is replacing the oldest expansion term stored in the buffer, which in this case was the input at $t = 11$. The weight for the 1st expansion term stored at $t = 11$ was large and $\mathbf{x}_t \approx \mathbf{x}_1$, therefore the truncation error reaches a maximum at $t = 111$. The effect of truncation error is completely eliminated by the delta correction feature in the SDC-KLMS algorithm.

Figure 27 shows the effect learning rate has on the mean squared error of both

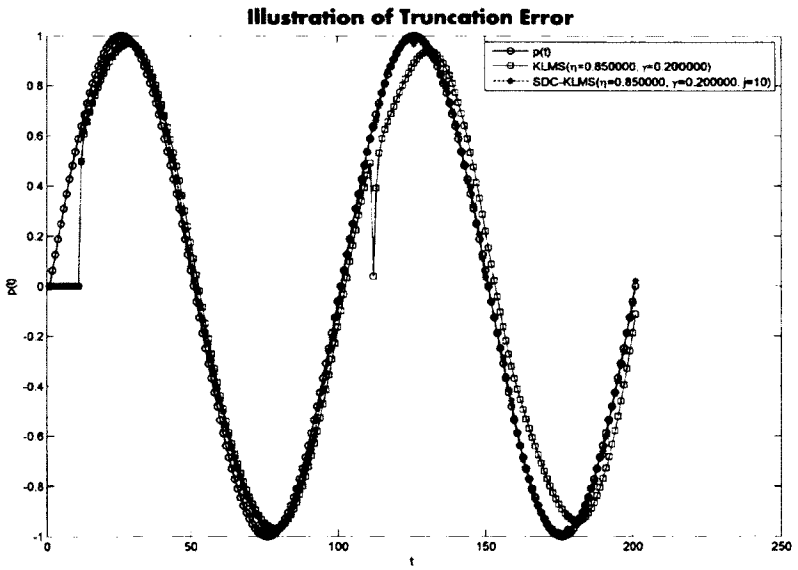


**Figure 26:** Truncation error is the most evident when t=111.

KLMS and SDC-KLMS in this particular case. The learning rate is an important consideration when looking at truncation error because it is the learning rate that determines the magnitude of the kernel expansion term weights. Figure 28 shows that in this particular case increasing the number of updated kernel expansion terms does little to improve the accuracy of SDC-KLMS.

The next test creates $p(t)$ by sampling $s(t) = \sin(4.34\pi t) + \mathcal{N}(0.0, 0.03)$ with a sampling frequency of 50Hz for a duration of 20 seconds. A small amount of normally distributed noise with $\mu = 0.0$ and $\sigma^2 = 0.03$ is added to each sample to avoid repeated inputs. The top panel in Figure 29 shows the time series $p(t)$ with blue circles and the predicted values of the time series produced by KLMS. The scale has been expanded to illustrate where the truncation error is the worst. The middle panel gives a color plot of the expansion term weights as a function of time. With KLMS once a sample is placed in the buffer its value does not change, as can be seen by the horizontal striping effect. The lower panel shows the sorted range of kernel evaluations for each expansion term and the current input. This plot is helpful for

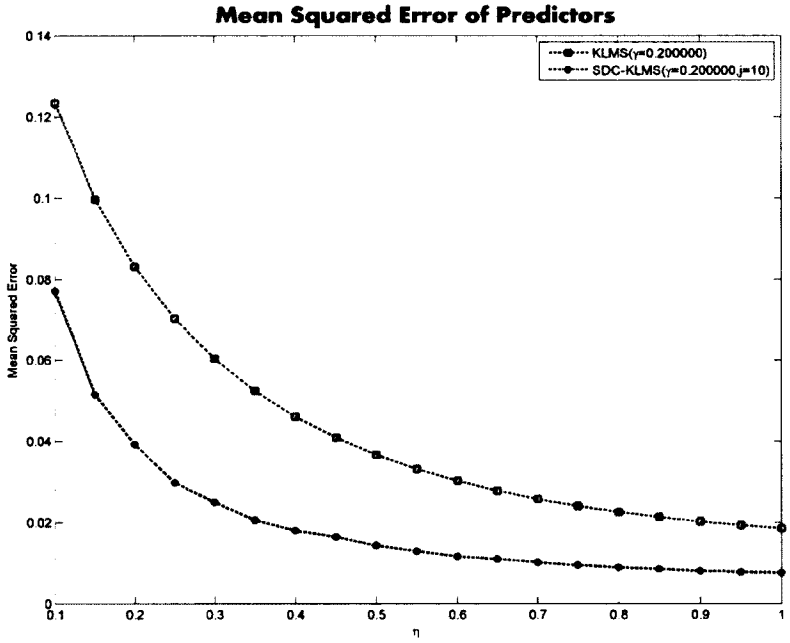


**Figure 27:** It is evident that the error gap between SDC-KLMS and KLMS is a trade off between low learning rates creating low truncation error but slow convergence, and high learning rates which create faster convergence but greater truncation error.

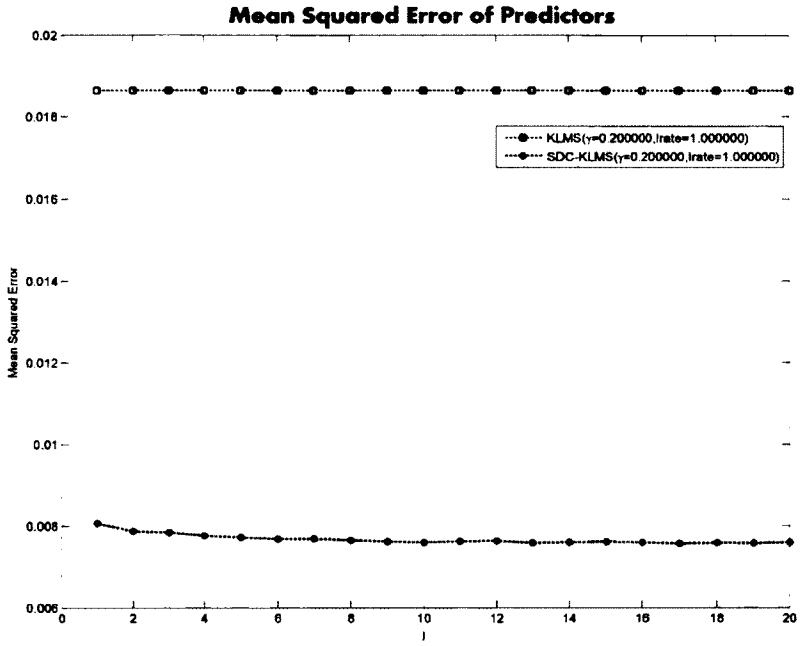


**Figure 28:** Increasing the number of updated kernel expansion terms does little to improve the accuracy of SDC-KLMS. This will not always be the case, especially in noisy highly dynamic environments.

deciding the kernel width parameter, $\gamma$. If $\gamma$ is set to too large a value, most kernel evaluations will evaluate to 0.0, and the kernel machine will not generalize well. If $\gamma$ is set to too small a value, most kernel evaluation will be close to 1.0, and the kernel machine will not distinguish between different inputs.

Figure 30 highlights the differences between KLMS and SDC-KLMS. The truncation error in the top panel is not visible as with KLMS. The middle panel in Figure 30 also illustrates the smoothing effect that the SDC-KLMS algorithm has on the expansion term weights in the buffer. By spreading the values of similar terms out, the effect of replacing a particular term is minimized with respect to the kernel machine output.

Figure 31 shows the error curve for the output of both SDC-KLMS and KLMS as a function of time. Even though SDC-KLMS compensates for truncation error, it does experience a small increase in error after for $t > 511$. When using this method for

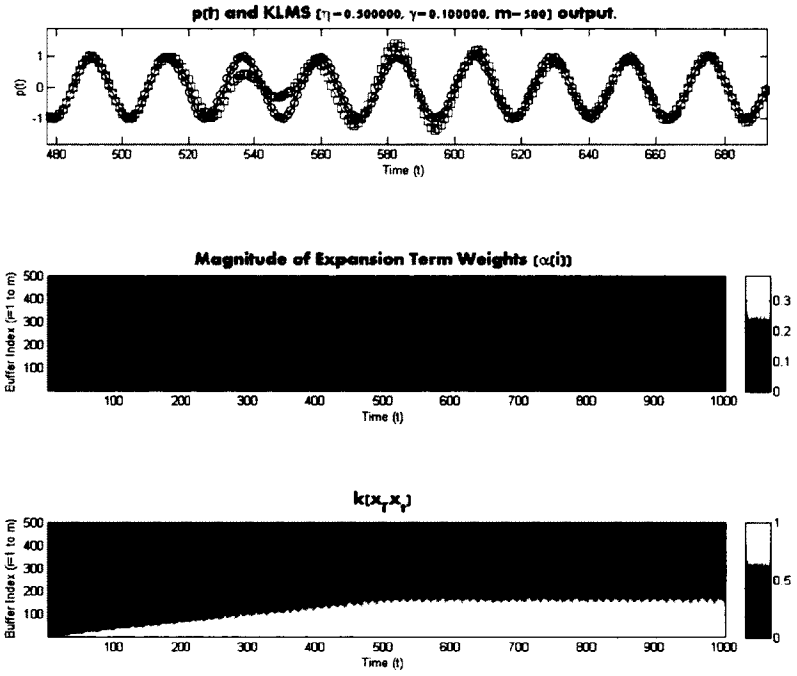


**Figure 29:** The top panel illustrates the truncation error is worse in the time series prediction. The middle panel shows the static nature of KLMS, and the buffer size becomes evident by the vertical shift in the magnitude of the expansion term weights. It can be seen that once old values start to get replaced in the kernel machine buffer, KLMS adjusts for this in the weights following the replacement. The bottom panel shows the values of the kernel evaluations for each input stored in the buffer. This is useful for setting the kernel width parameter.

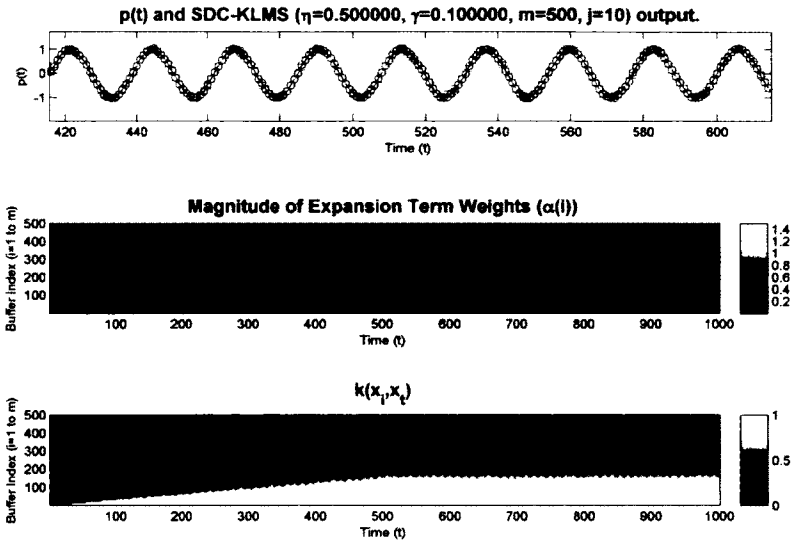


**Figure 30:** The smoothing effect on the weights of the kernel expansion terms is visible in the middle panel.

truncation compensation, the contribution of the old expansion term being replaced is added to the new expansion term in the kernel machine. The total number of terms is reduced by one, and that may have an effect on the accuracy of the kernel machine overall. The change in error for SDC-KLMS appears relatively constant rather than a large spike exhibited by KLMS.

### 5.5.2 Mackey-Glass Chaotic Time Series Simulation

The Mackey-Glass chaotic time series is often used in time series forecasting applications and is described by the differential equation given in Equation (74). I implemented a simulation of the differential equation in Simulink with the following parameters: $b = -0.1$, $a = 0.2$, $\tau = 30$, $s(0.0) = 1.2$, start time = 0.0, stop time = 6000, fixed time step, and solver = ode3. To each sample a small amount of normally distributed random noise was added with $\mu = 0.0$ and $\sigma^2 = 0.01$.

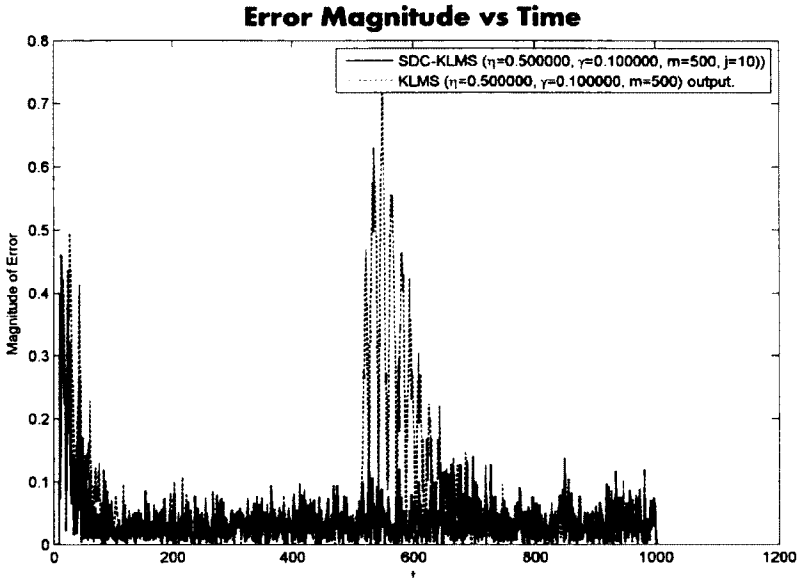


**Figure 31:** The absolute value of the error between the predicted value and the actual value of $\mathbf{p}(t+1)$ for each time step $t$. KLMS corrects for truncation error over time, while SDC-KLMS does not experience the truncation error it has one less expansion term to use for prediction.

$$\frac{d(s(t))}{dt} = -bs(t) + \frac{as(t-\tau)}{1+s(t-\tau)^{10}} \tag{74}$$

The Mackey-Glass time-series can exhibit periodic tendencies, as shown by the curve with blue circles the top panel in Figure 32. KLMS is shown with red squares and for $t > 510$ the truncation error starts to have a noticeable effect. KLMS then adapts to the truncation error and the transient error fades away. The error curve between KLMS and SDC-KLMS can be found in Figure 33. It has very similar characteristics to the noisy sine wave experiment from the previous section. This is significant because the Mackey-Glass time series is a benchmark dataset that is often used to test regression algorithms.

Figure 34 shows a interesting example of how great an effect truncation error can have on the performance of KLMS. The time series $\mathbf{p}_t$ is shown with the blue curve

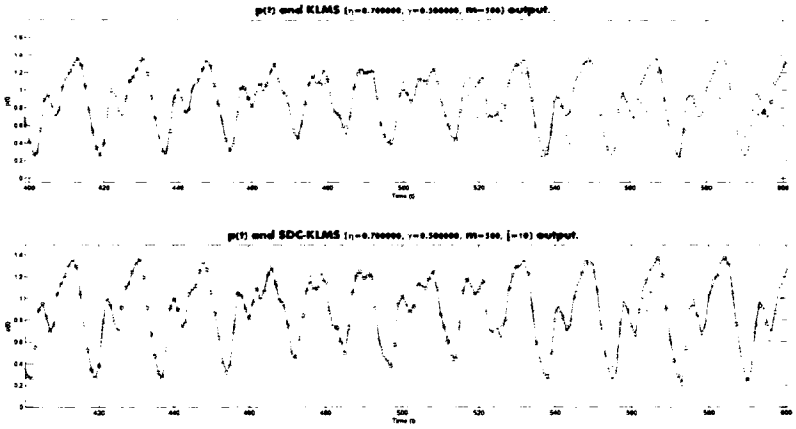

**Figure 32:** The top panel shows $\mathbf{p}(t)$ (blue circles) and the KLMS output, and the bottom panel shows $\mathbf{p}(t)$ (blue circles) and the SDC-KLMS output. Time scales have been adjusted to show when old expansion terms are replaced by new ones.

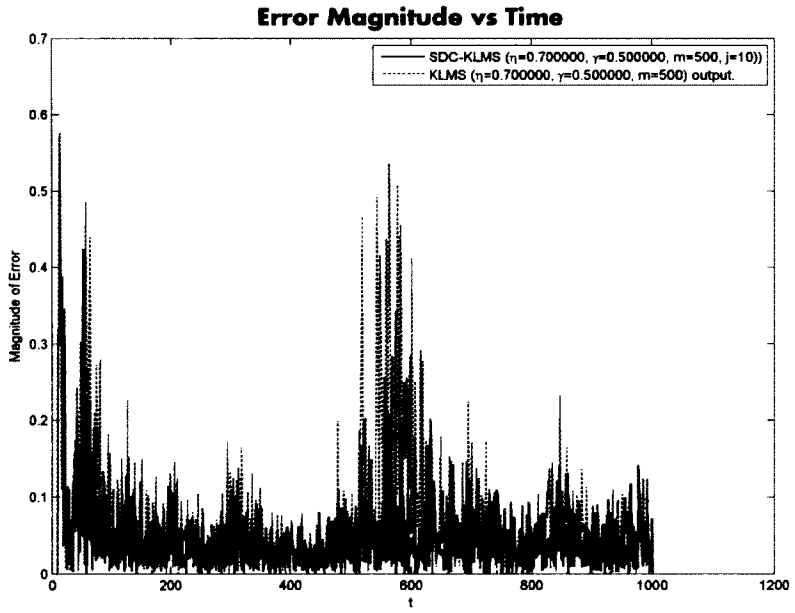


**Figure 33:** The absolute value of the error between the predicted value and the actual value of $\mathbf{p}(t+1)$ for each time step $t$. As before, KLMS corrects for truncation error (red curve) while SDC-KLMS does not experience the truncation error (blue curve).

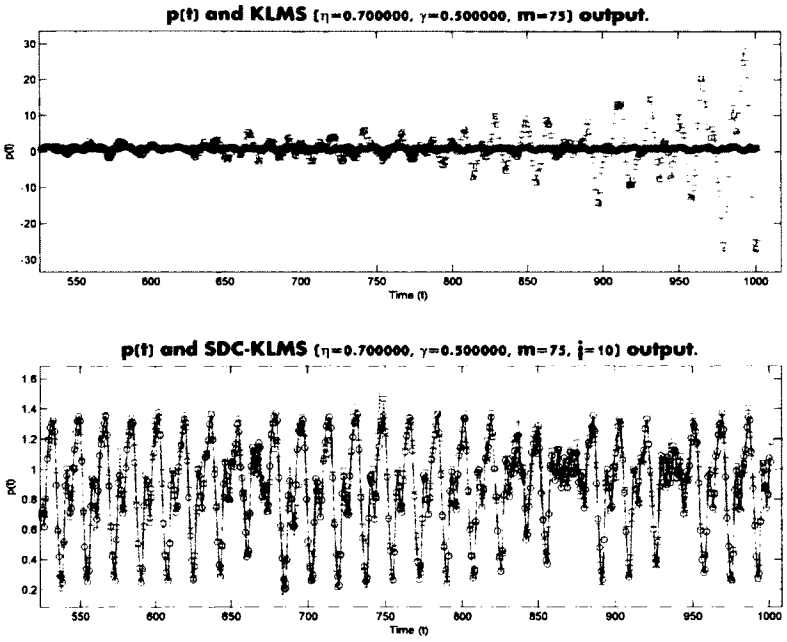


**Figure 34:** In this instance the top panel shows how KLMS becomes unstable when trying to predict the Mackey-Glass time series. The lower panel shows how the weight smoothing and delta correction features of SDC-KLMS keep the prediction function stable.

and circular markers. The top panel shows that the output of the KLMS predictor has turned unstable. This result is interesting because the learning rate is set to $\eta = 0.7$ which is less than the conservative bound of $\eta < 1.0$ [53] [51]. The bottom panel of Figure 34 shows how the SDC-KLMS algorithm does not experience the instability that KLMS does.

### 5.5.3 Phantom$^{TM}$ 3-Axis Trajectory Capture

This final experiment involves the use of KLMS and SDC-KLMS in the prediction of three dimensional trajectories from a Sensible Phantom$^{TM}$ haptic device. The three dimensional time series was captured from a haptic device at a sampling rate of 60Hz and is shown in Figure 35. The 3D haptic time series can be considered three separate time series, $\mathbf{p}_x$, $\mathbf{p}_y$, and $\mathbf{p}_z$. For this experiment a time embedding dimension of 31,

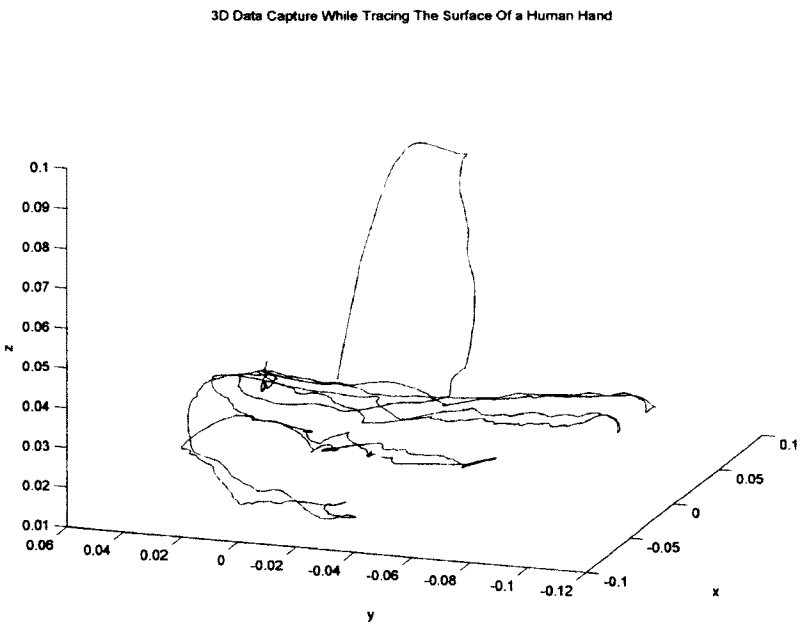


**Figure 35:** A 3D view of the captured trajectory used in our experiment.

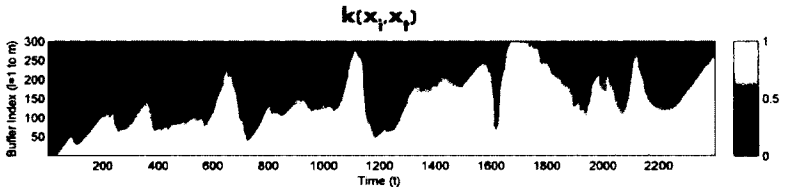


**Figure 36:** Kernel plot showing the distribution of kernel evaluations over time.

$\mathbf{x}_t = \left[\mathbf{p}_x\left(t-9\right), \mathbf{p}_y\left(t-9\right), \mathbf{p}_z\left(t-9\right) \cdots \mathbf{p}_x\left(t\right), \mathbf{p}_y\left(t\right), \mathbf{p}_z\left(t\right)\right]$ and $y_t = x\left(t-1\right)$ was used. By performing time embedding in this manner information is coupled from all three axis in the input of the kernel machine. The output of the kernel machine is predicting the next value for the x-axis coordinate. Three kernel machines would be required if we wished to predict all three axis coordinates.

There are some times where the input is common compared to all other expansion terms in the buffer. While other times the kernel evaluation is almost 0.0 across the entire buffer, this can be attributed to the motion of tracing the fingers on the hand and returning to the same point on the hand during the motion.

The error curves for this time series using KLMS and SDC-KLMS are almost identical as shown in Figure 37. This is an indication that truncation error is not playing a significant role in the error characteristics of either kernel machine. The kernel machines have the following parameterizations throughout our experiments: SDC-KLMS ($\eta = 0.2, \gamma = 25.0, m = 300, j = 10$) and KLMS ($\eta = 0.2, \gamma = 25.0, m = 300$). After running the experiment the mean squared prediction error for SDC-KLMS and KLMS was $1.86 \times 10^{-5} mm^2$ and $2.0318 \times 10^{-5} mm^2$ respectively. The fact that SDC-KLMS has a slightly better MSE for predicting the x-coordinate time series can be attributed to the extra $j$ weight updates at each iteration.

Referencing Equation (72), it is known that the truncation error will be the largest when two conditions are met. If the expansion term being replaced (at position $i$ in the buffer) has a large weight value and the current term being added to the buffer has a large kernel evaluation with the term being removed (i.e. for the RBF kernel used in our work, $k(\mathbf{x}_t, \mathbf{x}_i)$. These two conditions give us some insight into why our haptic experiment does not experience truncation error.

Figure 36 shows that at certain times the kernel evaluations across the entire buffer are close to 1.0. This kernel evaluation corresponds to slow motion where time series values are very similar to each other (i.e. $\uparrow k(\mathbf{x}_t, \mathbf{x}_i)$). The term weights are determined by the error that that term has with the prediction function and this error is lower for slow motions (i.e. $\downarrow \boldsymbol{\alpha}(i)$). For fast motion, the opposite is true, $\downarrow k(\mathbf{x}_t, \mathbf{x}_i)$ and $\uparrow \boldsymbol{\alpha}(i)$. These two trends will keep the truncation error low in this case.

These trends are subject to the size of the buffer used $m$, and the rate of change of the time series being predicted. Depending on the particular situation, it is likely to have severe truncation error in time series generated by haptic instruments and kinematic systems.

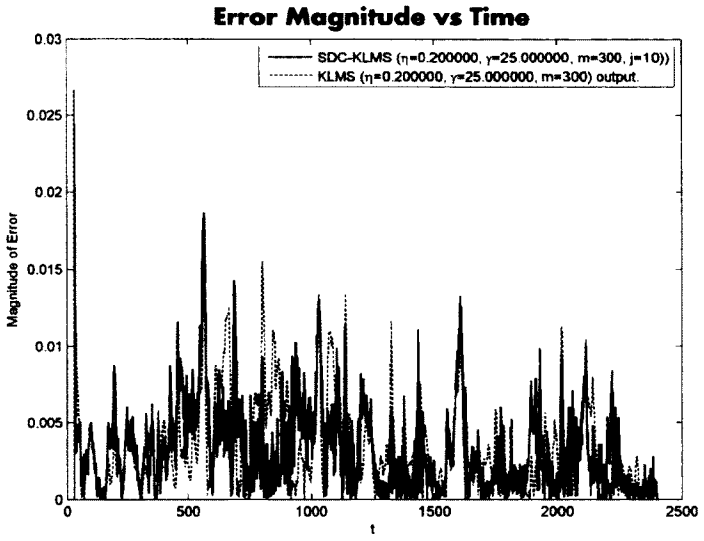


**Figure 37:** SDC-KLMS ($\eta = 0.2, \gamma = 25.0, m = 300, j = 10$) and KLMS ($\eta = 0.2, \gamma = 25.0, m = 300$)

## 5.6 Conclusions and Application Scope

There are two conflicting issues when using kernel machines in online systems requiring time series prediction. The first issue is the accuracy of the predictor in the system. The second concern in the computational load required to make a prediction.

Due to the online nature of the kernel least mean square algorithm, truncation error can be a significant issue for the accuracy of the kernel machine. The novel contribution presented in this chapter, the Smoothed Delta Corrected Kernel Least Mean Square (SDC-KLMS) algorithm, provides two mechanisms for a more accurate prediction function. The first mechanism combines compensation for truncation error, which we refer to as delta compensation. The second mechanism uses stochastic gradient descent using randomized indexes across all kernel expansion terms stored in the buffer to provide faster convergence of the predictor function. Truncation error is verified in these experiments using both simulated and benchmark data.

The SDC-KLMS algorithm can be applied in online intelligent systems with strict

finite memory requirements. This thesis has given much attention to computational complexity when considering online applications. Restricted memory resources are present especially in the area of micro-controllers and embedded computing environments. The SDC-KLMS algorithm is especially useful for online regression for embedded control and signal processing applications that are not tolerant to possible truncation error instabilities.

# Chapter 6

# Online Kernel Adaptation

Online processing of information requires not only that a system be computationally efficient, but also that also it can adapt to the changing requirements of its operating environment. It is the purpose of this chapter to look for an efficient but robust method for the adaption of a RBF kernel parameter within a kernel machine.

## 6.1 Chapter Outline

For preliminary material on the concept of time-embedding in the use of kernel machines for regression applications please refer to Section 2.7. Section 6.2 of this chapter will introduce some of the current work on kernel parameter adaptation from the point of view of kernel machines. Section 6.3 will introduce the idea of fuzzy logic based kernel parameter adaptation in terms of online kernel machines and data reconstruction in online environments. The experimental results of fuzzy logic based kernel adaptation are given in Section 6.4. Finally, Section 6.7 will give conclusions and insights on the results presented in this chapter.

## 6.2 Online Kernel Parameter Adaptation

The objective when tuning a kernel in an online fashion is to achieve an acceptable balance between processing time and desired accuracy. The kernel function can be viewed as a similarity measure between inputs and/or support vectors in the kernel expansion. The Gaussian kernel has a universal approximating capability, and is numerically stable [51]. When using the Gaussian kernel in a kernel machine, the choice of kernel width is the most important decision regarding the accuracy of the kernel machine.

There are three common approaches to tuning a kernel parameter. First, given a batch of training data, one can tune the parameter by trial and error until the desired accuracy is achieved. This can be time consuming, inaccurate, and may not be appropriate for an online situation. The second approach involves cross-validation techniques. Cross-validation uses a dataset that is split up for testing and training and can provide an unbiased measure of the accuracy of the kernel machine. It can avoid over fitting to the dataset, but it is also a time consuming process. The third method involves the use of Bayesian inference to determine the model parameters. In [51], the authors describe using kernel recursive least-squares and Bayesian methods to determine kernel model parameters. This approach also requires a dataset to operate on, and requires iterative optimization.

Recent research in the area of online adaptation of kernel parameters can be found in [75–77]. For instance, in the work of Ucak and Oke [75], the authors used adaptive kernel parameter tuning in adaptive PID control tuning. The authors achieved positive results in tuning the kernel parameter using gradient descent, while using an online least square support vector machine (LSSVM) model in their PID system.

In the work of Singh and Principe [77], the kernel parameters within a time delay neural network are adapted using information theoretic cost functions. The results

of the authors work show that their system converges more quickly by adaptation of the kernel parameter during training.

Further, Sun [76], takes a novel approach in kernel parameter tuning by using a curve fitting method to avoid evaluating the kernel machine in the adaptation phase. The work presented in this thesis takes a similar approach by not relying on the output of the kernel machine to adapt the kernel parameter. Instead of using a distance metric in RKHS, as in [76], the solution in this chapter uses input space statistics and fuzzy logic adaptation of the kernel function parameter.

## 6.3 Fuzzy Logic Kernel Parameter Adaptation

This approach is unique because it makes use of the simple observation that similar data vectors should produce a kernel evaluation closer to 1.0 than 0.0. In a similar fashion, dissimilar vectors should produce a Gaussian kernel evaluation closer to 0.0 than 1.0. Figure 38 illustrates this online approach to adaptation of the Gaussian kernel parameter using both input statistics and the similarity measure in RKHS. The kernel parameter is tuned using a fuzzy logic inference system (FIS).

The use of the moving average filter in Figure 38, allows the sensitivity of the adaptation process to be tuned to more recent observations or to older observations. Throughout this research, the following relationship is used to produce the moving average, $\mathbf{x}_{AVE} = 0.05\mathbf{x}_t + 0.95\mathbf{x}_{AVE}$. The choice of weighting for the moving average was based on experimental performance and is problem dependent. The adaptation of $\gamma$ is performed at each time step by the update equation, $\gamma = \gamma + c\Delta_\gamma$.

The Mamdani FIS was implemented using the Fuzzy Logic Toolkit within Matlab, and has the following input membership functions:

- *muchLessThan* (trapmf): [-1 -1 -0.8 -0.6]

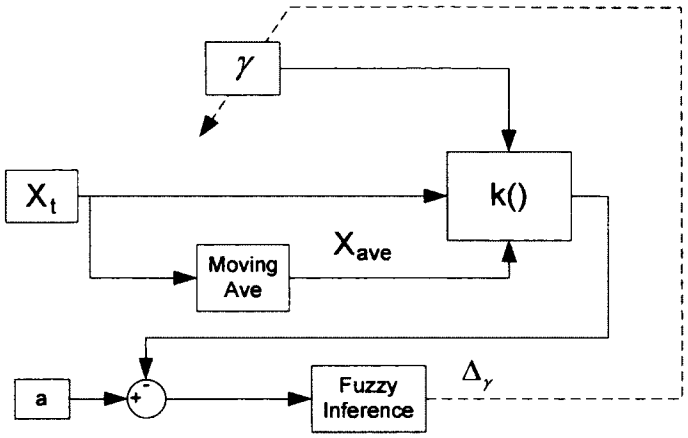


**Figure 38:** The adaptation procedure used in this work. By using a moving average procedure in the input space of the filter, it is possible to adapt the kernel parameter based on multiple previous inputs.

- *lessThan* (trimf): [-0.8 -0.4 -0.05]
- *equal* (trimf): [-0.1 0 0.1]
- *greaterThan* (trimf): [0.05 0.4 0.8]
- *muchGreaterThan* (trapmf): [0.6 0.8 1 1]

The output membership functions are as follows:

- *decreaseGammaMuch* (trapmf): [-1 -1 -0.8 -0.6]
- *decreaseGamma* (trimf): [-0.8 -0.4 0]
- *keepConstant* (trimf): [-0.2 0 0.2]
- *increaseGamma* (trimf): [0 0.4 0.8]
- *increaaeGammaMuch* (trapmf): [0.6 0.8 1 1]

The fuzzy linguistic rules are the following:

- If input is *equal* then output is *keepConstant.*
- If input is *lessThan* then output is *increaseGamma.*
- If input is *muchLessThan* then output is *increaseGammaMuch.*
- If input is *greaterThan* then output is *decreaseGamma.*
- If input is *muchGreaterThan* then output is *decreaseGammaMuch.*

The rules are combined using the centroid method and the crisp output is scaled by a tunable factor, $c$, to produce $\gamma$. The choice of using fuzzy logic is based on two reasons. First, fuzzy logic systems are able to deal with uncertainty, which is common in online systems. Second, the parameter adaptation of the kernel parameter can be defined in terms of human language rules, simplifying the tasks of system designer.

## 6.4 Experiments with Data Reconstruction

All experiments follow the *time embedding* process and notation described in Section 2.7. The fuzzy logic controller was implemented combination with a KLMS [53] filter for evaluation. For evaluating the data recovery procedure, a time-series dataset was used in two ways. First, we measured the accuracy of the KLMS filter with and without the adaptive kernel parameter and also with a static kernel parameter. Then the KLMS filter was evaluated with the adaptive kernel on the same dataset, but dropouts were introduced. Figure 39 illustrates both cases, with and without dropouts.

## 6.5 Sinusoid with Noise

Equation (75) describes the first time series used to evaluate this algorithm.

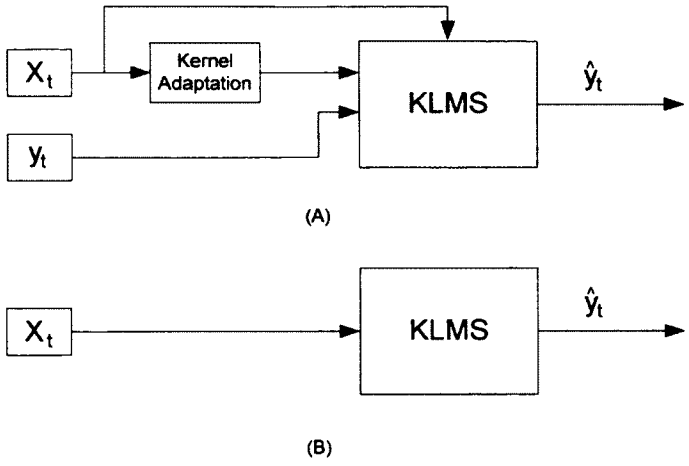


**Figure 39:** (A): $\hat{y}_t$ represents the output of the KLMS filter when data arrives within a specified time deadline. (B): When a dropout occurs, the kernel parameter is not adapted. $\mathbf{x}_t$ contains the previous time-series values but $y_t$ is not available, therefore the kernel parameter is not adapted.

$$\mathbf{p}_t = 0.2\,(rand\,(0,1) - 0.5) + \sin\left(\frac{2\pi t}{1000}\right) + 1.0 \tag{75}$$

Figure 40 shows how the kernel parameter, $\gamma$ is adapted over time. The average set value, $a$, is reached and the kernel evaluation oscillates around the set point due to the oscillation of the input vector.

Figure 41 illustrates the case of an instantaneous change in the nature of the input. At $t = 5000$, the amplitude of the sinusoid was increased by a factor of 1.3. Figure 41 shows how the approach adapts to the changing input to maintain the same kernel evaluation.

To simulate the effect of dropouts a probability of 0.1 per time step of a dropout occurring was used. If a dropout did occur, a duration of 1 to "Max Duration" time steps was randomly selected. Both randomizations were from a uniformly distributed distribution. Table 11 gives the results of function estimation with simulated communication dropouts. It can be seen that the effect of dropout does have an effect on

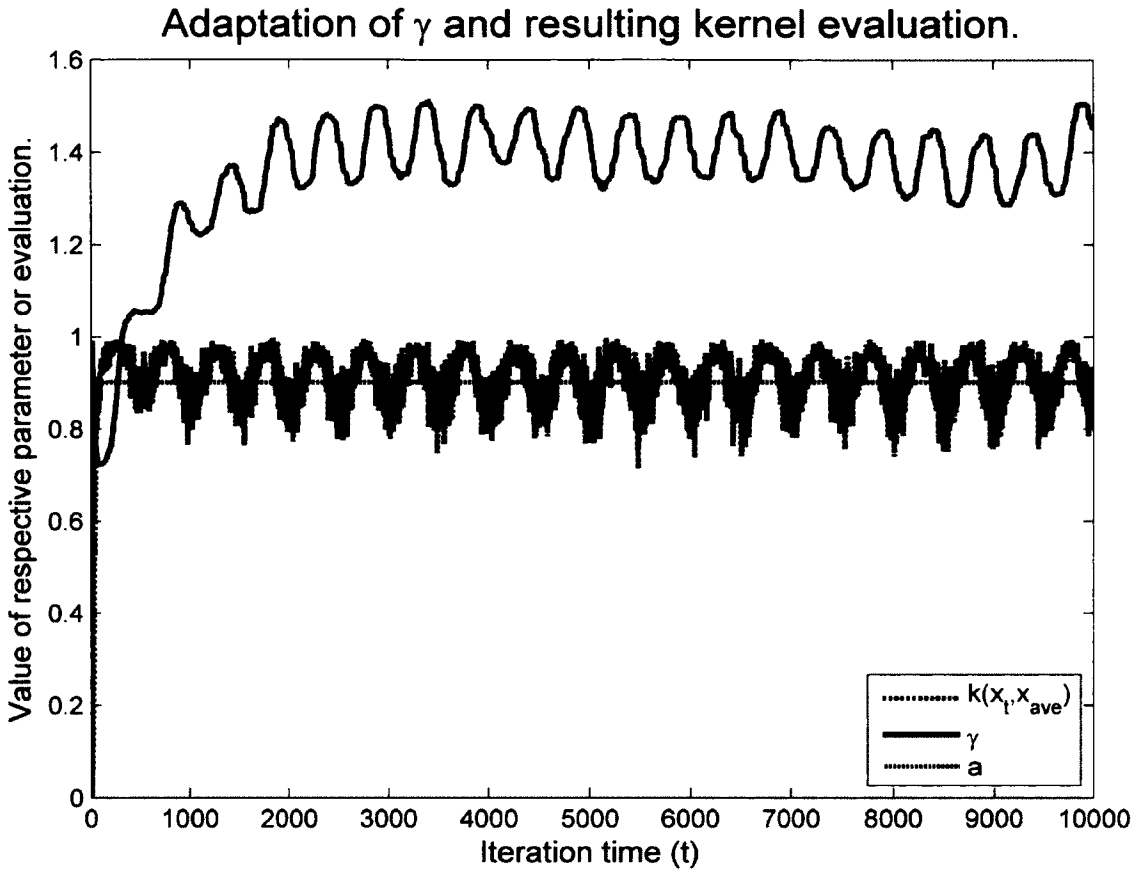


**Figure 40:** The time progression of the kernel parameter adaptation technique. $\gamma$ was initially set to 1.0.

the accuracy, but if the duration is low enough, the error can be minimized.

**Table 11:** Experiment parameters were as follows: $m = 400$, $\eta = 0.8$, $n = 7$, $c = 0.01$. A parallel experiment with constant $\gamma = 4.0$ was carried out with no dropouts, and the error incurred was 0.0859.

| ***Sinusoidal Error Rates*** | |
|---|---|
| Max Duration | Mean Error |
| No Dropout | 0.0653 |
| 7 | 0.0681 |
| 8 | 0.0763 |
| 10 | 0.1030 |

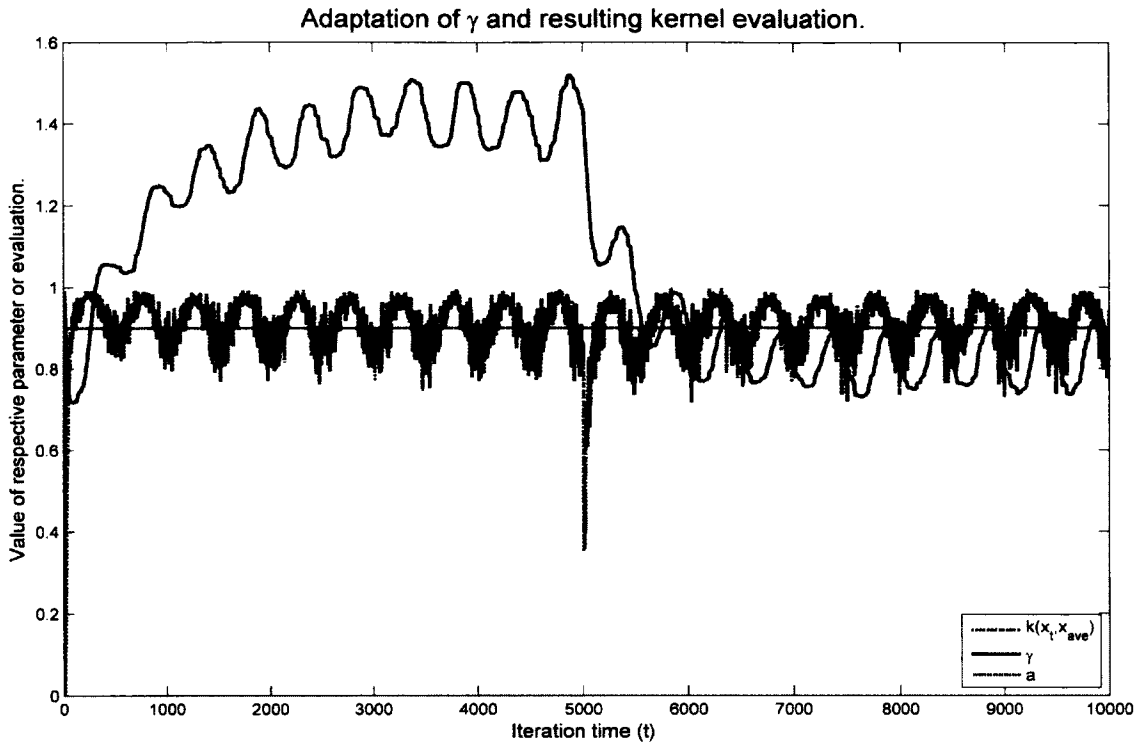


**Figure 41:** The time progression of the kernel parameter adaptation technique. $\gamma$ was initially set to 1.0 and at time 5000 the amplitude of the sin wave was adjusted. The FIS adapts the kernel parameter to compensate for the change in the input. The system recovers within 50 time steps.

An identical experiment was conducted using a constant value of $\gamma = 4.0$ with no data dropouts, the mean error incurred was 0.0859. The benefit of using fuzzy logic adaptation is evident because even in the presence of dropouts, the error performance is superior when using fuzzy logic to adapt the kernel parameter. Only when the maximum dropout duration reaches 10 time steps did the error performance become worse than the constant $\gamma$ case.

## 6.6 Sensable 6DOF Phantom Haptic Device

The second test scenario involved the capture of position data from a 6 degree-of-freedom (DOF) haptic device. The Sensable Phantom (TM) was used, and the sampling rate used to capture the trajectories was 60 Hz. The 3D position represented the end tip of the device, and a tracing motion was used on top of a person's hand to capture common trajectories that would occur when interacting with a soft material.

Figure 42 illustrates the captured trajectory of the x and y coordinates only. This gives a top down view of the path. Three KLMS filters were used in parallel with adjustment of the kernel parameters done individually for each kernel machine.

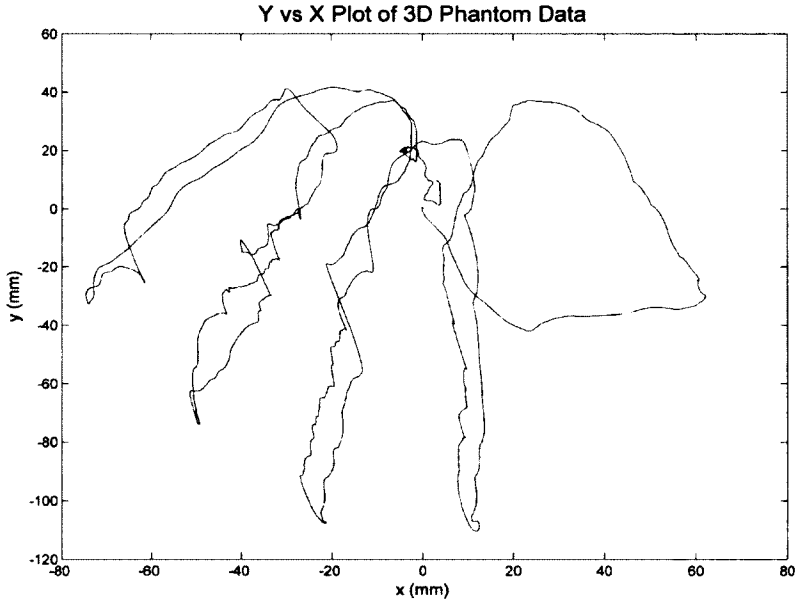


**Figure 42:** A x-y view of the 3D trajectory capture from the Sensable Phantom.

Figure 43 shows the adaptation of $\gamma$ for the x axis trajectory only. Due to the high variation of the trajectory at some instances of the experiment, the value of gamma was adjusted to a very small value. The use of input statistics is a disadvantage in two particular cases. In the first case for time sequences where the position varies widely from one time step to the next, the fuzzy logic controller cannot adapt the

kernel parameter fast enough. In the second case the position does not change from one time step to the next, and therefore the adaptation has no effect.

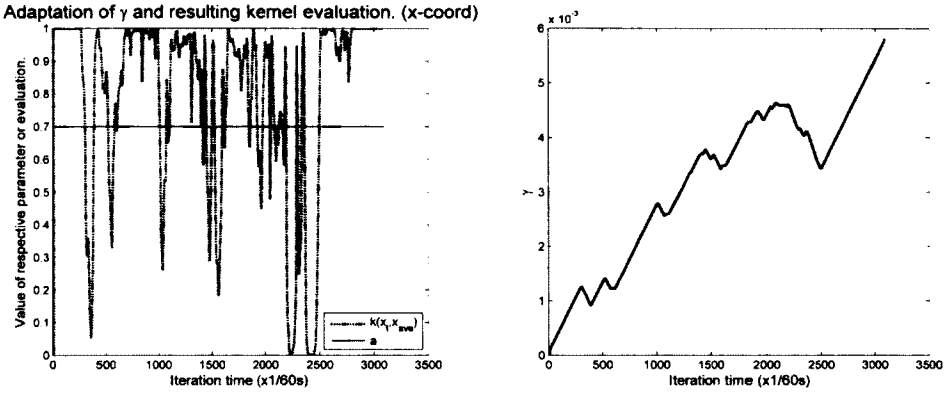


**Figure 43:** Extreme swings in the time embedded input vectors make it difficult to adapt the $\gamma$ parameter. Equally distorting are constant values in the input, causing the kernel function to evaluate to 1.0 regardless of the parameter $\gamma$.

**Table 12:** Experiment parameters for the x, y, and z axis were as follows: $m = 100$, $\eta = 1.0$, $n = 10$, and for the adaptation on the x-axis, $c = 0.00001$.

| ***Phantom 3D Error Rates*** | | | |
|---|---|---|---|
| Max Duration | Mean Error x (mm) | Mean Error y (mm) | Mean Error z (mm) |
| No Dropout | 0.8203 | 1.2632 | 1.1102 |
| 4 | 0.9831 | - | - |
| 6 | 1.8492 | - | - |
| 15 | 3.9344 | - | - |

Aggressive tuning of the kernel parameter has little effect when the input exhibits great amplitude changes but will have extreme effects on the accuracy of the filter. As with the previous experiment, when dropouts are introduced the accuracy is impacted only slightly until the duration of the dropout increases to 6 time steps or higher. Table 12 shows that this adaptive approach achieves accuracies in the range of 1 mm during the experiment. Although Table 12 does not illustrate the advantage of fuzzy

logic tuning of the kernel parameter, it does show that data reconstruction is possible using the proposed approach.

## 6.7 Conclusions and Application Scope

In this chapter is a novel method for tuning the width of the Gaussian kernel function in a kernel machine for online applications. The use of fuzzy logic allows for human reasoning to be placed into the control of the capacity of the kernel machine.

The results from this chapter show that fuzzy logic can be successfully applied to the problem of RBF kernel parameter tuning. Fuzzy logic allows linguistic expressions to define the nature of the tuning and is easy for system designers build an adaptation scheme for their intelligent systems. Fuzzy logic adaptation is also successful in cases of switching data as shown in Figure 41. Input space statistics is a simple tool for evaluating the performance of the adaptation procedure, but it has degraded performance in the case of extremely noisy data as shown by the the results from the haptic device experiment. The introduction of dropouts in the data does have a negative effect on the accuracy of the kernel machine. But having the ability to adapt the model to the nature of the input is in line with a true online learning scenario, such is the case of data recovery in tele-haptic and tele-operation environments.

The novel contribution of fuzzy logic control of online adaptation of the RBF kernel parameter is fundamentally different from the research conducted in [75–77]. The use of fuzzy logic in this chapter is similar to the research of Sun [76] in that the output of the kernel machine is not required for adaptation of the parameter. The difference from [76] is the use of input space statistics to avoid computationally expensive kernel expansion optimization. The use of fuzzy logic allows the integration of linguistic rules for kernel machine specification.

Application area for fuzzy logic adaptation of the RBF kernel parameter is in

online signal processing and control. The input space statistics used in this chapter are time based and cannot be used in an offline learning scenario. Time based processing of input data is useful for signal processing applications such as non-linear filtering, non-linear control, and non-linear online regression applications.

# Chapter 7

# An Online Kernel Machine for Anomaly Detection and Mode Tracking

Kernel based algorithms for multivariate density estimation, mode tracking and anomaly detection have provided some effective solutions to problems in the area of pattern recognition and analysis. Some examples are: object tracking using the mean-shift [78], the family of $\nu$-SVND [62] and NORMA [40] algorithms. The purpose of this chapter is to develop a training algorithm that solves the quadratic programming problem of a $\nu$-SVND while at the same time addressing the problems of online adaptability and training efficiency.

## 7.1 Chapter Outline

Preliminary material on $\nu$-SVND the NORMA can be found in Sections 2.5.2 and 2.6 respectively. Section 7.2 introduces the application area of the algorithm presented in this chapter. Section 7.3 describes the Active Set (AS) algorithm for solving the QP problem with considerations of online use. Next, results for two different experimental settings are explained in Section 7.4. Finally, in Section 7.5 an iterative mean shift equation is given that can find the maximum of a kernel density estimator.

## 7.2 Application Setting

The use of online methods for anomaly detection and mode tracking have the requirement of computationally efficient execution. In the work of Schölkopf et al. [62] the extension of the traditional support vector machine approach was used to develop an algorithm for anomaly detection, specifically novelty detection. The use of a separating hyperplane between the origin and the data points in Reproducing Kernel Hilbert Space (RKHS) allows for a non-linear separation in the input space. Section 2.4 and 2.5.2 give more details on RKHS and support vector novelty detection (SVND), respectively.

An online extension of the work in [62] was provided by Kivinen et al. [40]. Instead of solving a quadratic programming problem to train the kernel machine, they make use of *stochastic gradient descent* of a regularized risk functional. The two primary drawbacks to this approach are the following:

- The use of stochastic gradient descent is error driven and there is no balance between function complexity and points lying outside the hyperplane.
- The use of regularization in the risk functional yields an estimator that favors newer samples over older ones [40]. This technique has been shown to produce a biased solution in other kernel machine implementations [51].

In an online system data arrives in a streaming fashion, while in an offline system, data is acquired and then processed in a batch fashion. The active set algorithm makes use of a similarity measure between previous data points and the current one to update all weights in the kernel machine. The use of an internal kernel matrix means that the storage requirement will grow in $O(m^2)$, where $m$ is the total number of vectors stored in the kernel machine buffer.

## 7.3 The Active Set Kernel Machine Algorithm

The goal of the active set algorithm is to incorporate newly introduced data into the kernel machine while still making positive progress in the optimization of the quadratic programming objective function. The Radial Basis Function kernel is used exclusively in this chapter. The use of chunking was explored extensively by Osuna et al. [79], and Platt [56]. Chunking optimizes a subset of variables in the QP problem at a time. Sequential minimal optimization (SMO) is an extreme example of chunking where only pairs of variables are optimized.

The development of the AS algorithm is similar to the SMO algorithm in that only pairs of variables are optimized with each iteration of the algorithm. Equation (76) states the QP problem that must be minimized to train the support vector novelty detector. The minimal solution will yield a maximum margin hyperplane that separates the data from the origin in RKHS.

### 7.3.1 Pair-wise Optimization

Novelty detection using SVMs was described in Section 2.5.2. The QP problem is revisited in this chapter with Equation (76). Equations (77) and (78) give the constraints that are applied to the QP problem. The box constraint in Equation (77) restricts that maximum and minimum values that can be assigned to expansion term weights.

$$Z(\alpha) = \frac{1}{2} \sum_{i,j=1}^{m} \alpha(i)\, \alpha(j)\, K_{i,j} \tag{76}$$

$$0 \leq \alpha(i) \leq \frac{1}{\nu m} \tag{77}$$

$$\sum_{i=1}^{m} \alpha(i) = 1 \tag{78}$$

Depending on the value of $\boldsymbol{\alpha}(i)$, there are three important distinctions that can be made about vectors that are retained in the kernel machine buffer. After optimization the following will apply to all vectors in the kernel machine buffer:

- $\boldsymbol{\alpha}(i) = 0$: $\mathbf{x}_i$ is not a support vector in the kernel expansion, and does not need to be included during the evaluation of the kernel expansion ($f(\mathbf{x}_i) > \rho$).
- $0 < \boldsymbol{\alpha}(i) < \frac{1}{\nu m}$: $\mathbf{x}_i$ sits exactly on the hyperplane in RKHS. This particular vector is referred to as a non-bound support vector. ($f(\mathbf{x}_i) = \rho$).
- $\boldsymbol{\alpha}(i) = \frac{1}{\nu m}$: $\mathbf{x}_i$ is outside the hyperplane and is referred to as a bound support vector ($f(\mathbf{x}_i) < \rho$).

The significance of these three types of vectors in the kernel machine buffer will play an important role in the optimization of the quadratic program. Since only pairs of variables are optimized in any one iteration, these two variables are referred to as $\alpha_a = \boldsymbol{\alpha}(a)$ and $\alpha_b = \boldsymbol{\alpha}(b)$. The change in notation is merely for convenience. Equation (79) separates $\alpha_a$ and $\alpha_b$ from the summation term in Equation (76).

$$\begin{aligned} Z(\alpha) = {} & \tfrac{1}{2}\alpha_a^2 K_{a,a} + \tfrac{1}{2}\alpha_a \left[ \sum_{i \neq a,b}^{m} \alpha_i K_{a,i} \right] + \tfrac{1}{2}\alpha_b^2 K_{b,b} \\ & + \tfrac{1}{2}\alpha_b^2 K_{b,b} + \tfrac{1}{2}\alpha_b \left[ \sum_{i \neq a,b}^{m} \alpha_i K_{b,i} \right] + \tfrac{1}{2} \sum_{i,j \neq a,b}^{m} \alpha_i \alpha_j K_{i,j} \\ & + \tfrac{1}{2}\alpha_a \alpha_b K_{a,b} + \tfrac{1}{2}\alpha_b \alpha_a K_{b,a} \end{aligned} \tag{79}$$

The following substitutions can be made:

$$\begin{aligned} K_{a,b} &= K_{b,a} \\ \chi_a &= \tfrac{1}{2}\left[\sum_{i\neq a,b}^{m} \alpha_i K_{a,i}\right] \\ \chi_b &= \tfrac{1}{2}\left[\sum_{i\neq a,b}^{m} \alpha_i K_{b,i}\right] \\ C &= \tfrac{1}{2}\sum_{i,j\neq a,b}^{m} \alpha_i \alpha_j K_{i,j} \end{aligned} \tag{80}$$

Yielding the a simplified form for Equation (76):

$$Z(\alpha) = \frac{1}{2}\alpha_a^2 K_{a,a} + \frac{1}{2}\alpha_b^2 K_{b,b} + \alpha_a \chi_a + \alpha_b \chi_b + \alpha_a \alpha_b K_{a,b} + C \tag{81}$$

Equation (81) is written only in terms of the variables $\alpha_a$ and $\alpha_b$, with all other variable being held constant. The equality constraint given in Equation (78), allows $\alpha_b$ to be written in terms of $\alpha_b$ (Equation (82)).

$$\alpha_b = 1 - \sum_{i\neq a,b}^{m} \alpha_i - \alpha_a \tag{82}$$

Since the QP objective function can be written fully in terms of $\alpha_a$, the derivative of $Z(\alpha)$ is taken with respect to $\alpha_a$. Equation (83) gives the derivative of $\alpha_b$ in terms of $\alpha_a$. Equation (84) takes the derivative of Equation (81) while using Equation (83) to keep the entire expression in terms of a single variable $\alpha_a$.

$$\frac{\partial(\alpha_b^2)}{\partial \alpha_a} = -2\left(1 - \sum_{i\neq a,b}^{m} \alpha_i - \alpha_a\right) \tag{83}$$

$$\frac{\partial Z(\alpha)}{\partial \alpha_a} = \alpha_a K_{a,a} - \left(1 - \sum_{i\neq a,b}^{m} \alpha_i - \alpha_a\right) K_{b,b} + \chi_a - \chi_b + \left(1 - \sum_{i\neq a,b}^{m} \alpha_i - 2\alpha_a\right) K_{a,b} \tag{84}$$

By setting the derivative of the objective function to zero, the minimum of the objective function can be found. These are shown in Equations (85) and (85).

$$\alpha_a K_{a,a} - K_{b,b} + K_{b,b} \sum_{i \neq a,b}^{m} \alpha_i + \alpha_a K_{b,b} + \chi_a - \chi_b + K_{a,b} - K_{a,b} \sum_{i \neq a,b}^{m} \alpha_i - 2\alpha_a K_{a,b} = 0 \quad (85)$$

$$\alpha_a = \frac{K_{b,b}\left(1 - \sum_{i \neq a,b}^{m} \alpha_i\right) - K_{a,b}\left(1 - \sum_{i \neq a,b}^{m} \alpha_i\right) - \chi_a + \chi_b}{K_{a,a} + K_{b,b} - 2K_{a,b}} \quad (86)$$

Finally, using the RBF kernel as the kernel function, all kernel evaluations across the diagonal of the Gram matrix $K$ are 1.0. Therefore Equation (86) can be further simplified to Equation (87).

$$\alpha_a = \frac{1}{2}\left(1 - \sum_{i \neq a,b}^{m} \alpha_i\right) + \frac{\chi_b - \chi_a}{2\left(1 - K_{a,b}\right)} \quad (87)$$

Once two vectors are chosen in the kernel machine buffer (indices $a$ and $b$) to have their $\alpha$ weights optimized, Equation (87) is used to directly solve for $\alpha_a$ and Equation (82) is used to recover $\alpha_b$.

The question still remains as to how the index values for $a$ and $b$ are determined. Osuna's theorem states that for positive progress to be made in the optimization problem, only vectors in the kernel machine that violate the Karush-Kunn-Tucker (KKT) conditions for optimality should be optimized [56]. The KKT violators can be identified by testing the vectors stored in the kernel machine buffer to see if any of the previously described non-support vectors or support vectors violate the below equivalences:

$$
\begin{aligned}
f(x_i) < \rho &\Leftrightarrow \alpha_i = \tfrac{1}{\nu m} \\
f(x_i) > \rho &\Leftrightarrow \alpha_i = 0 \\
f(x_i) = \rho &\Leftrightarrow 0 < \alpha_i < \tfrac{1}{\nu m}
\end{aligned}
\tag{88}
$$

Following a similar procedure to the work of Schölkopf et al. [62] for determining when a vector violates one of the KKT conditions described in Equation (88), a KKT violation can be detected by the use of the boolean expression described by Equation (89).

$$
\left(\left(f(\mathbf{x}_{index}) - \rho\right)\boldsymbol{\alpha}_{index} < 0\right) \vee \left(\left(\rho - f(\mathbf{x}_{index})\right)\left(\frac{1}{\nu m} - \alpha\left(i\right)\right) > 0\right) \tag{89}
$$

### 7.3.2 The Active Set (AS) Algorithm

It is assumed that data arrives in an online fashion at time index $t$. The AS algorithm works in a similar fashion to the modified SMO algorithm presented in [62], by optimizing two variables at once. The AS algorithm trains the kernel machine in three novel ways.

The first contribution is the margin parameter $\rho$ is calculated as the average of all non-bound support vectors. In the offline batch setting SMO assumes that any single non-bound support vector can be used to recover $\rho$. This technique is only true when the QP problem has been fully solved. In the case of SMO the value of $\rho$ gets updated with each optimization step and it converges to its true value. This approach is not suitable for an online environment. The QP objective function changes with each time step as a new data vector is added to the kernel machine buffer. Taking the average of each non-bound support vector prevents the value of $\rho$ from changing dramatically.

The second novel contribution is the use of a similarity measure when iterating over the set of points that violate the KKT conditions (this set is considered the active set). The RBF kernel evaluation can be viewed as a similarity between two vectors in the input space. All kernel evaluations between the input vector and all vectors stored within the kernel machine buffer are performed for the optimization of the QP problem. These values can also be used to impose and ordering of all KKT violating points. Looking at Equation (87) it can be seen that the value of $\alpha_a$ will be larger when choosing a closer vector index for $\alpha_b$ than one that is further away. Then when solving for $\alpha_b$ using Equation (82), this value will have a smaller value to maintain the equality constraint of the QP objective function. This will help to enforce sparsity in the number of bound and non-bound support vectors stored in the buffer. There are multiple assignments of $\alpha_b$ and only one assignment of $\alpha_a$ per time step, allowing $\alpha_a$ to become a support vector and at the same time promoting smaller weight values for vectors already in the buffer.

Finally, the third contribution is a result of the fact that the calculation of $\rho$ is only done at the start and the end of a full iteration. Contrasting to the modified version of SMO from [62], $\rho$ is calculated after every pair-wise optimization. The computational complexity will be examined further in Section 7.4.

The AS algorithm is presented below in Algorithm 5. The AS algorithm operates in two stages, whereby the first stage assigns an initial value to $\alpha(t)$ based on the evaluation of $f_{t-1}(\mathbf{x}_t)$ and then adjusts all other $\alpha$ values to account for the increase of vectors in the kernel machine buffer. The second stage identifies KKT violations and performs the pairwise optimization step between the input and all KKT violating vectors.

Algorithm 6 describes the pairwise optimization of all KKT violating vectors. Once the input vector has been added to the kernel machine buffer at time $t$, it's $\alpha$ weight is initialized based on $f_{t-1}(\mathbf{x}_t)$. Then it is treated like all other vectors in the

**Algorithm 5** The Active Set Kernel Machine Algorithm

Initialize:

- Kernel Machine Buffer, $X \leftarrow \emptyset$
- Expansion Term Buffer, $A \leftarrow \emptyset$
- Kernel Matrix, $K \leftarrow \emptyset$
- $\rho \leftarrow \emptyset$
- Number of stored vectors, $m \leftarrow 0$

**while** $t < \inf$ **do**
  $\mathbf{x}_t \leftarrow$ from input stream.
  $X \leftarrow X \cup \mathbf{x}_t$
  **if** $t = 0$ **then**
    $\boldsymbol{\alpha}(0) \leftarrow 1.0$
    $K_{1,1} \leftarrow k(\mathbf{x}_t, \mathbf{x}_t)$
    $\rho \leftarrow 1.0$
  **else**
    **if** $f_{t-1}(\mathbf{x}) < \rho_{t-1}$ **then**
      $\boldsymbol{\alpha}(t) \leftarrow \frac{1}{\nu(m+1)}$
    **else if** $f_{t-1}(\mathbf{x}) > \rho_{t-1}$ **then**
      $\boldsymbol{\alpha}(t) \leftarrow 0.0$
    **else**
      $\boldsymbol{\alpha}(t) \leftarrow \frac{1}{2\nu(m+1)}$
    **end if**
    $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha}\left(\frac{m}{m+1}\right)$
    $m \leftarrow m + 1$
    **for** $i \leftarrow 1$ to $m$ **do**
      $K(i, m) \leftarrow k(\mathbf{x}_i, \mathbf{x}_m)$
      $K(m, i) \leftarrow k(\mathbf{x}_i, \mathbf{x}_m)$
    **end for**
    index_keys $\leftarrow$ sort($K(1 \cdots m, m)$)
    optKKT($\boldsymbol{\alpha}$,$X$,$\nu$,$\gamma$,index_keys)
  **end if**
**end while**

kernel machine buffer. If it is identified as a KKT violator it will be assigned index $a$ for pairwise optimization. If it is not identified as a KKT violation then the closest KKT violator to the current input will be assigned index $a$.

**Algorithm 6** optKKT($\boldsymbol{\alpha}$,$X$,$\nu$,$m$,$\gamma$,index_keys)

1: $\rho \leftarrow$ calculate_rho($\boldsymbol{\alpha}$,$X$,$\gamma$)
2: KKT Violation List, $VKKT \leftarrow \emptyset$
3: **for** i= 1 to $m$ **do**
4: index $\leftarrow$ index_keys($i$)
5: $\mathbf{x}_{index} \leftarrow X\,(index)$
6: **if** Equation (89) evaluates to **TRUE** . **then**
7: $VKKT \leftarrow VKKT \cup (i)$
8: **end if**
9: **end for**
10: a $\leftarrow VKKT \cup (1)$
11: **for** i= 2 to $m$ **do**
12: b $\leftarrow VKKT \cup (i)$
13: $\boldsymbol{\alpha}\,(a)$,$\boldsymbol{\alpha}\,(b) \leftarrow$ by equations 87 and 82.
14: **end for**

**Algorithm 7** calculate_rho($\boldsymbol{\alpha}$,$X$,$\nu$,$m$,$\gamma$)

1: $c \leftarrow 0$
2: $\rho \leftarrow 0$
3: **for** i= 1 to $m$ **do**
4: **if** $0 < \boldsymbol{\alpha}\,(i) < \frac{1}{\nu m}$ **then**
5: $\rho \leftarrow \rho + f(\mathbf{x}_i)$
6: $c \leftarrow c + 1$
7: **end if**
8: **end for**
9: return $\rho/c$

## 7.4 Experiments

The first experiment that was performed to evaluate the AS algorithm was on 2-dimensional data points. The following points were used to train the novelty detector $\mathbf{x}_{t=1} = [0.0, 0.0]$, $\mathbf{x}_{t=2} = [1.0, 0.0]$, $\mathbf{x}_{t=3} = [-1.0, 0.0]$, $\mathbf{x}_{t=4} = [0.0, 1.0]$,

$\mathbf{x}_{t=5} = [0.0, -1.0]$, $\mathbf{x}_{t=6} = [2.0, 2.0]$. This small set of points was chosen so that the AS-algorithm could be tested visually. A more realistic data set will be tested later in this chapter.

Figure 44 illustrates the resulting novelty detectors, ND1 and ND2. The two top panels (a and b) within the figure show the resulting kernel expansion evaluations. Green squares represent non-bound support vectors, while red x's represent bounded support vectors. The value of the margin parameter $\rho$ only depends on non-bound support vectors, so those particular vectors have an effect on $\rho$. Panels c and d show the resulting QP objective function for the novelty detectors ND1 and ND2 respectively. The red line represents the equality constraint that any solution must enforce. Panel c shows that the optimal point for the QP occurs when $\mathbf{x}_{t=6} = [2.0, 2.0]$ becomes a bound support vector (due to box constraints in the QP) and $\mathbf{x}_{t=1} = [0.0, 0.0]$ becomes a non-bound support vector. The choice of $\nu$ determines the location of the box constraints. In panel d, both vectors are non-bound support vectors. The choice of RBF kernel parameter $\gamma$ affects the curvature of the QP objective function.

The experiment illustrated in Figure 45 was performed by alternate sampling of two separate, two dimensional distributions, $\mathbf{x}_{t=ODD}$=[$\mathcal{N}(\mu = -2.0, \sigma = 0.5)$, $\mathcal{N}(\mu = -2.0, \sigma = 0.5)$] and $\mathbf{x}_{t=EVEN}$=[$\mathcal{N}(\mu = 2.0, \sigma = 0.5)$,$\mathcal{N}(\mu = 2.0, \sigma = 0.5)$]. A total of 100 samples were generated starting with $t = 1$ and then alternating between $\mathbf{x}_{t=EVEN}$ and $\mathbf{x}_{t=ODD}$.

Panels a and c, and panels b and d in Figure 45 represent two novelty detectors each trained with the same parameters for $\nu$ and $\gamma$ respectively. In one case the novelty detector was trained using the AS algorithm (panel a and c), and in the other case the novelty detector was trained using Matlab's quadprog() optimizer (panel b and d).

It can be seen that there are some differences between the resulting solutions.

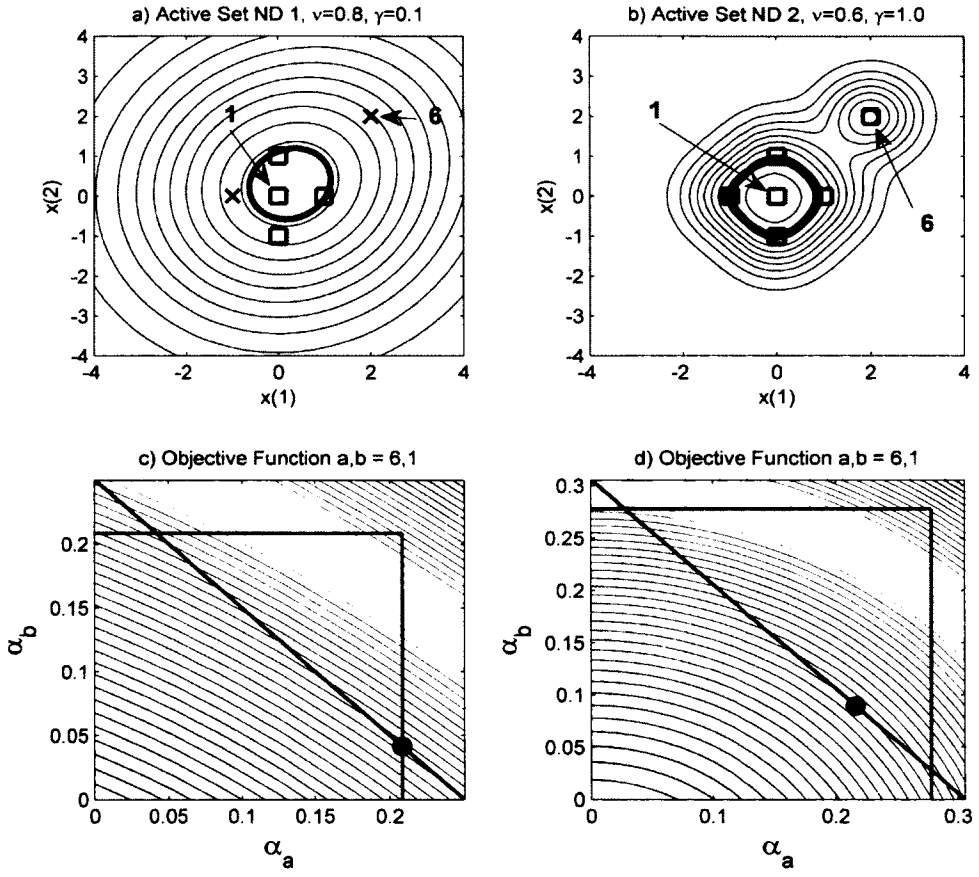


**Figure 44:** a) A contour plot of a novelty detector, ND1, trained using the AS algorithm. Data point 1 and 6 were added at time steps $t = 1$ and 6 respectively. b) A contour plot of a novelty detector, ND2, trained using the Active Set algorithm. Data point 1 and 6 were added at time steps $t = 1$ and 6 respectively. c) Due to parameterization of $\nu$ and $\gamma$ point 6 becomes a bounded support vector, while point 1 is a non-bounded support vector. d) Due to parameterization of $\nu$ and $\gamma$ both points 6 and 1 become a non-bound support vectors.

Matlab's quadprog() optimizer will provide a precise solution to the quadratic program. The resulting number of non-bound support vectors in the active set solution are greater than the QP solution.

Figure 46 illustrates a similar result to that of Figure 45. In this case the kernel parameter was increased, yielding similar novelty detectors. It is evident that the

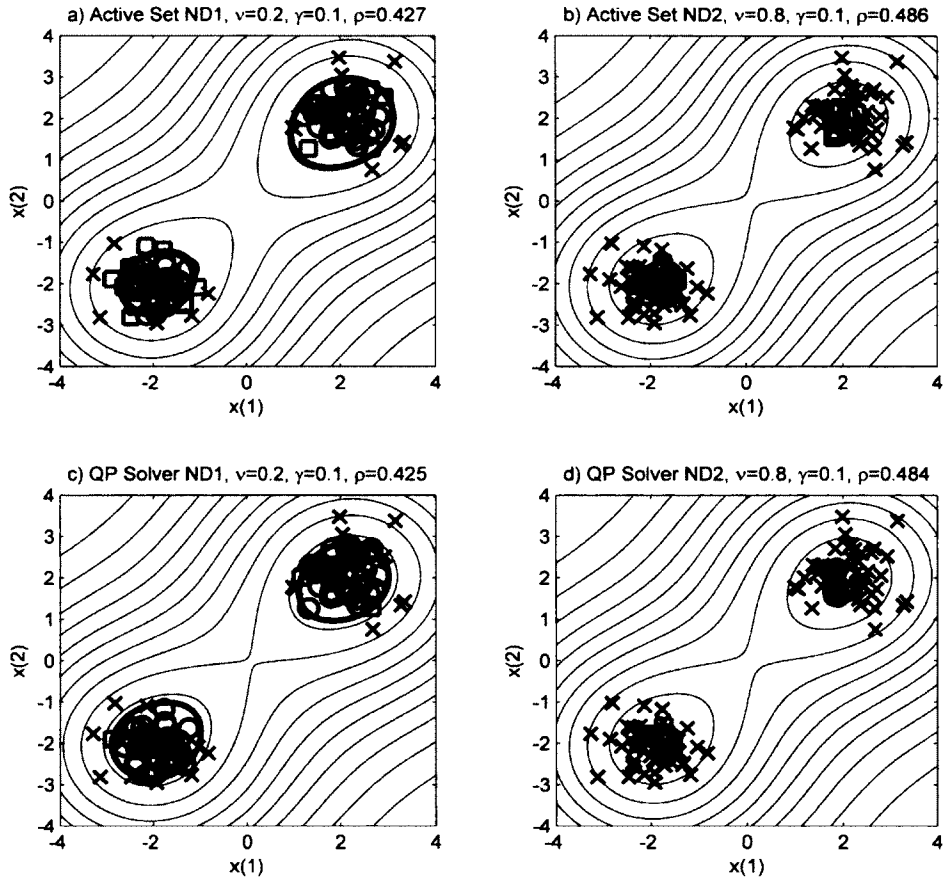


**Figure 45:** Comparisons between two different support vector novelty detectors. Panels a and b: In this setting the value of $\gamma$ remains fixed while $\nu$ is adjusted. Training was accomplished by the AS algorithm. Panels c and d: The same data set and parameters were used as in panels a and b. Training was accomplished by using Matlab's quadprog() quadratic programming solver. $\rho$ was calculated in the same manner as in the AS algorithm.

result of averaging non-bound support vectors during optimization will yield more non-bound support vectors.

Table 13 shows that the solution obtained by the AS algorithm and the QP optimized solution both tend to enforce the same $\nu$ property as in the work of Schlkopf et al. [62]. The $\nu$ property sets bounds for both the number of non-bound and bound

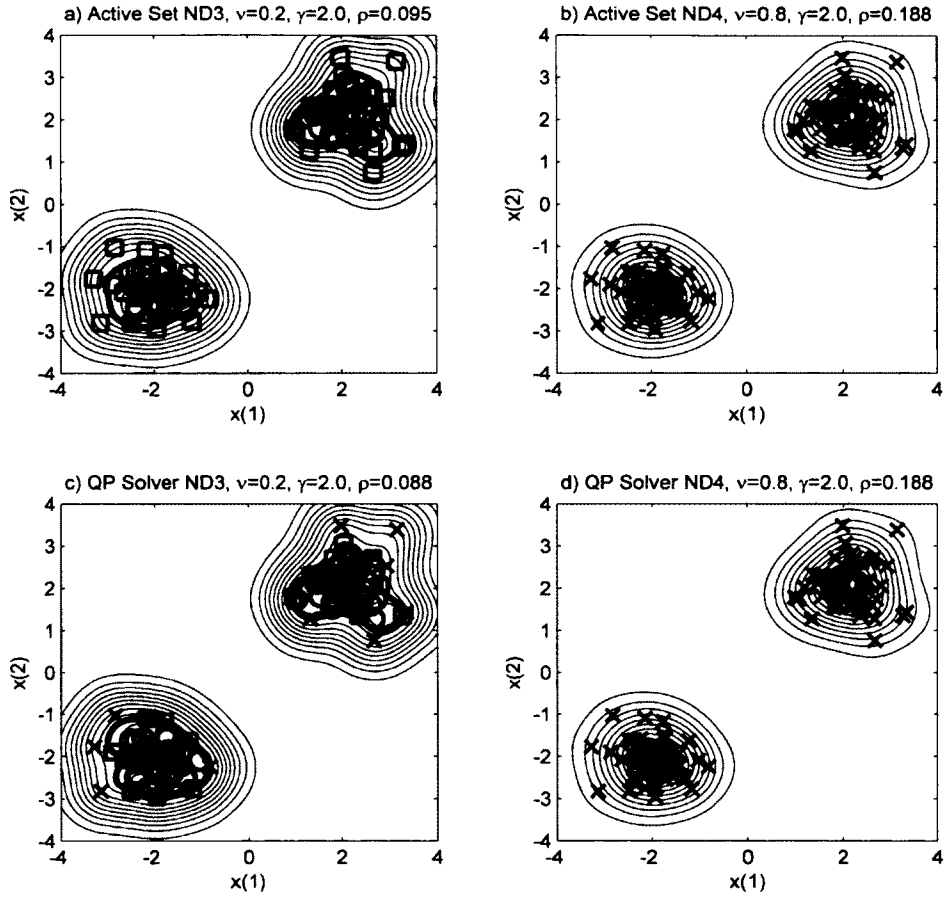


**Figure 46:** Similar to the plots found in Figure 45, but in this instance the width of the kernel function was made smaller by increasing $\gamma$. The resulting novelty detector from the AS algorithm produces a kernel machine with more non-bound support vectors.

support vectors, and the number of outlier data points across all observed data. The set of outliers is defined in the following sense, $O = [i | i \in \{1 \cdots t\} \wedge f_t(x_i) < \rho]$.

In the case of the number of outliers, the $\nu$-bound places an upper limit on the outliers that will occur on the training set. In the case of the number of support vectors, the $\nu$-bound represents an upper limit on the total number of support vectors (bound and non-bound) that will be produced from training.

**Table 13:** Summary of results for novelty detectors illustrated in Figures 45 and 46.

| Novelty Detector | $\nu$-bound | Number of SVs | Number of outliers ($\|O\|$) |
|---|---|---|---|
| AS ND1 ($\nu = 0.2, \gamma = 0.1$) | 20 | 30 | 20 |
| AS ND2 ($\nu = 0.8, \gamma = 0.1$) | 80 | 90 | **83** |
| AS ND3 ($\nu = 0.2, \gamma = 2.0$) | 20 | 52 | 18 |
| AS ND4 ($\nu = 0.8, \gamma = 2.0$) | 80 | 89 | 78 |
| QP Solver ND1 ($\nu = 0.2, \gamma = 0.1$) | 20 | 21 | 19 |
| QP Solver ND2 ($\nu = 0.8, \gamma = 0.1$) | 80 | 82 | 79 |
| QP Solver ND3 ($\nu = 0.2, \gamma = 2.0$) | 20 | 34 | **21** |
| QP Solver ND4 ($\nu = 0.8, \gamma = 2.0$) | 80 | 82 | **81** |

The results in Table 13 are in line with Figures 45 and 46. The AS algorithm operates in an incremental fashion, incorporating new values at every time step. The total number of support vectors produced tends to be higher than that of an optimal QP solution because KKT violator pairs are only optimized once per time step. In the case of anomaly detection, an outlier is any input pattern, $\mathbf{x}_i$, that $f(\mathbf{x}_i) < \rho$. The outlier count also appears to break through the $nu$-bound in the case of AS ND2. Since the value of $\rho$ is calculated by averaging across all non-bound support vectors, the value of $\rho$ will converge to a value that will enforce the $\nu$-bound as more values are added to the kernel machine buffer. The behavior of tracking an input distribution is desirable for an algorithm that processes data in an online manner. Table 14 provides evidence of the convergence of the algorithm to the $\nu$-bound by running the exact same experiment as above, but this time with 300 samples. Any cases where the $\nu$-bound is broken in the QP trained novelty detectors is due to round off errors and machine precision. Matlab's quadprog() function can return values very close to 0.0. Any $\boldsymbol{\alpha}(i)$'s that were close to 0.0 or close to $\frac{1}{\nu m}$ were set to 0.0 or $\frac{1}{\nu m}$ respectively.

Figure 47 illustrates how using the AS algorithm archives better computational complexity than the use of a specialized quadratic programming solver. General purpose QP solvers will scale in $O(m^3)$ [56,62] computational complexity, where $m$ is

**Table 14:** Summary of results for novelty detectors illustrated in figures 45 and 46.

| Novelty Detector | $\nu$-bound | Number of SVs | Number of outliers ($\|O\|$) |
|---|---|---|---|
| AS ND1 ($\nu = 0.2, \gamma = 0.1$) | 60 | 68 | 60 |
| AS ND2 ($\nu = 0.8, \gamma = 0.1$) | 240 | 250 | 239 |
| AS ND3 ($\nu = 0.2, \gamma = 2.0$) | 60 | 135 | **71** |
| AS ND4 ($\nu = 0.8, \gamma = 2.0$) | 240 | 244 | 230 |
| QP Solver ND1 ($\nu = 0.2, \gamma = 0.1$) | 60 | 62 | 59 |
| QP Solver ND2 ($\nu = 0.8, \gamma = 0.1$) | 240 | 242 | **241** |
| QP Solver ND3 ($\nu = 0.2, \gamma = 2.0$) | 60 | 69 | 59 |
| QP Solver ND4 ($\nu = 0.8, \gamma = 2.0$) | 240 | 242 | **241** |

the number of variables being optimized. The AS algorithm scales in computational complexity according to $O(mk)$ where m is the number of variables being optimized and k is the number of KKT violators. Algorithm 7 line five shows that the kernel expansion is evaluated for each KKT violator. Since the number of KKT violators will be less than the number of vectors being optimized the computational complexity of the AS algorithm will scale between $O(m)$ and $O(m^2)$. Interestingly this is the same computational complexity as the SMO algorithm [56]. The training is done on a step by step basis in the As algorithm, so the number of KKT violators will usually remain low, giving a computational complexity closer to $O(m)$ than to $O(m^2)$. Figure 47 shows the near linear computational complexity of the AS training algorithm.

Execution of kernel machines will be faster than any other density estimator that does not enforce a sparsification technique. The use of a Parzen window estimator is a good example of the online computational complexity of incorporating each input vector into the density estimate. The computational complexity of evaluating a Parzen window estimator will scale in $O(t)$ computational complexity. In other words, with each time step, one more vector is added to the kernel machine buffer that must be evaluated in the kernel expansion expression. The key advantage of kernel machines is that only a subset of observed data is used in the kernel expansion

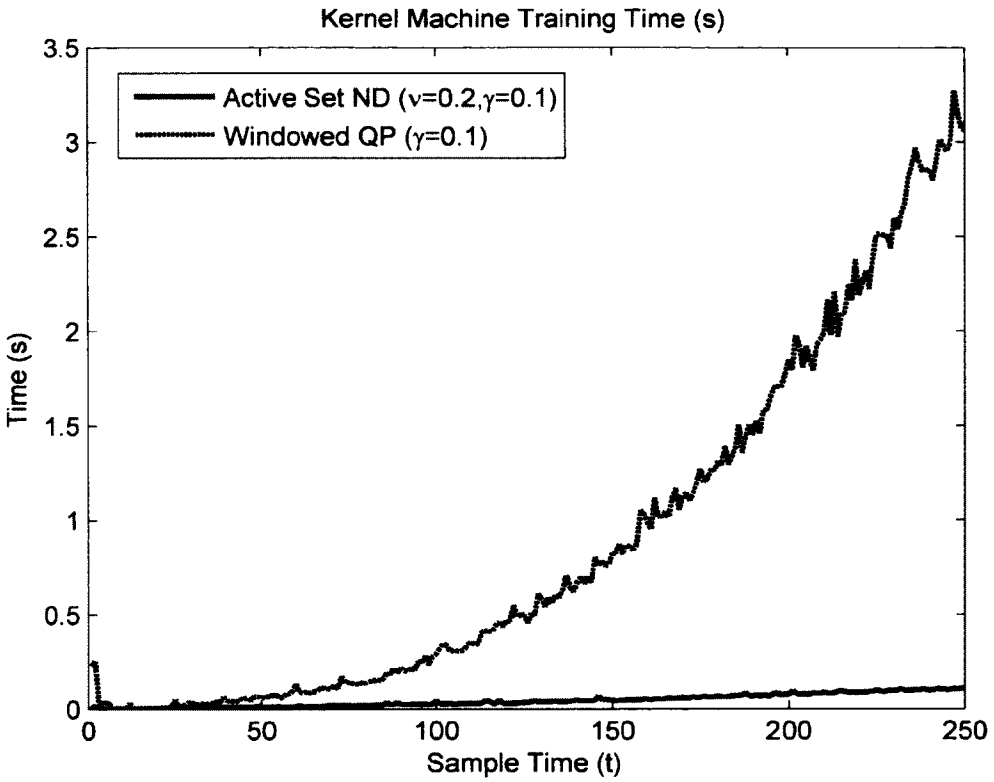


**Figure 47:** Training using Matlab's quadprog() quadratic programming solver was accomplished by solving a successively larger QP problem at each time step. The AS algorithm executes faster because it incorporates each new input into the existing kernel machine.

evaluation. The computational complexity of evaluating a kernel machine trained using the active set approach is $O(n)$, where $n$ is the total number of bound and non-bound support vectors.

Figure 48 compares the execution performance of both a Parzen window estimator and a novelty detector trained using the AS algorithm. Both algorithms operate with linear computational complexity. The kernel machine novelty detector is subject to the *Representer Theorem* [1]. The sparsification employed by the kernel machine results in better linear computational complexity due to hidden constants. Run-time efficiency is a highly desirable characteristic in online kernel machine applications.

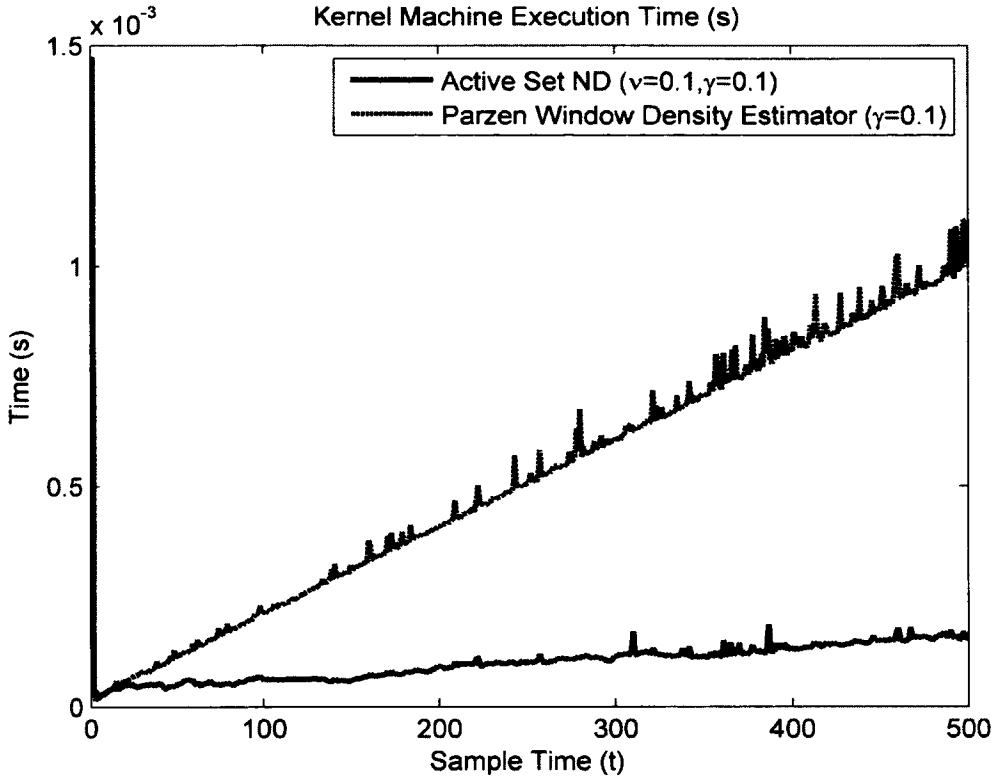


**Figure 48:** Evaluation time was compared to a Parzen Window kernel density estimator (KDE). A Parzen Window estimator is equivalent to $\nu$-SVND when $\nu$=1.0, and therefore every input becomes a bound support vector. The resulting AS trained $\nu$-SVND performs better because only support vectors make up the expansion evaluation.

## 7.5 Mode Tracking

A kernel machine used as a novelty detector can also be used as a density estimator and mode tracker for the underlying probability distribution that the input data is drawn from. It is important to point out that the parameterization of the kernel function is important when using a $\nu$-SVND to estimate a density function on the input data. Figure 46 panels a and c illustrate that depending on parameterization, $f(\mathbf{x})$ may not accurately reflect the probability distribution that the input data is being drawn from. Panels a and c use $\nu = 0.2$ which will place an upper bound on

the number of outliers in the training set to 20% of the training set size. The number of support vectors are lower bounded to 20% of the training set size. The kernel width is too small in this case, and the effect is that $f(\mathbf{x})$ does not resmble the actual probability distribution. Panels b and d show that increasing $\nu = 0.8$ will allow more support vectors to be included into the kernel expansion and then the shape of $f(\mathbf{x})$ is more representative of the actual probability distribution.

In the work of Comaniciu and Meer [80] and Comaniciu et al. [78] the authors explore using the mean shift for input space mode detection and object tracking in a sequence of images respectively. In their work for mode detection Comaniciu and Meer define the kernel density estimator using a Parzen window estimator [80]. Equation (90) illustrates the complete formulation of a Parzen window kernel density estimator. Equation (91) gives the changes made to Equation (90) to fit within the use of kernel machines.

$$\hat{f}(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right) \tag{90}$$

In Equation (90), the number of kernel functions used in the density estimate is defined by the parameter $n$. The kernel function evaluation is based on a euclidean metric that is scaled by a bandwidth parameter, $h$ raised to the power of the dimensionality, $d$, of the input. Equation (91) simplifies the kernel expansion by weighting each kernel function by $1/m$. While the contributions of each kernel function will no longer satisfy certain statistical properties, the modification only changes the scale of the density estimate. The shape of the density estimate is still the same and since the mean shift procedure seeks the maximum of the density estimate, removal of the bandwidth parameter will have no effect on the result. In the case of Equation (92), the constant weighting on each kernel function is replaced by an input dependent weighting. This is the familiar form of the kernel expansion that we use in our AS

algorithm.

$$\hat{f}(x) = \frac{1}{m}\sum_{i=1}^{m} k(x, x_i) \tag{91}$$

$$f(x) = \sum_{i=1}^{m} \alpha(i)\, k(x, x_i) \tag{92}$$

Equations (93) and (94) give the original expression for the mean shift and the modified version that can be used in the kernel expansion form of the active set algorithm. In Equation (93) the term $g()$ is equivalent to $-k'()$. When using the RBF kernel in this example the -ve sign and the effect of the parameter $\gamma$ cancel out in the numerator and the denominator. In [80] the constant weighting on each kernel term is factored out of the mean shift procedure. Equation (94) has to account for the variable weight terms.

$$x_{MS} = \frac{\sum_{i=1}^{n} x_i g(x_{MS}, x_i)}{\sum_{i=1}^{m} g(x_{MS}, x_i)} \tag{93}$$

$$x_{ASMS} = \frac{\sum_{i=1}^{n} x_i \alpha(i)\, k(x_{ASMS}, x_i)}{\sum_{i=1}^{m} \alpha(i)\, k(x_{ASMS}, x_i)} \tag{94}$$

The modified version of the mean shift procedure is illustrated in Figure 49. Initial points were chosen at $x_{neg} = [-4.0 - 4.0]$ and $x_{pos} = [4.04.0]$. Equation (94) was repeatedly applied until there was little change in the either density maximum. The large square markers settle at the two maximum points of the density estimate.

The primary benefit of the modified version of the mean shift is that the update procedure only needs to use non-zero $\alpha$ weights. In other words, as with the density estimate itself, the search for the maximum of the density function only depends

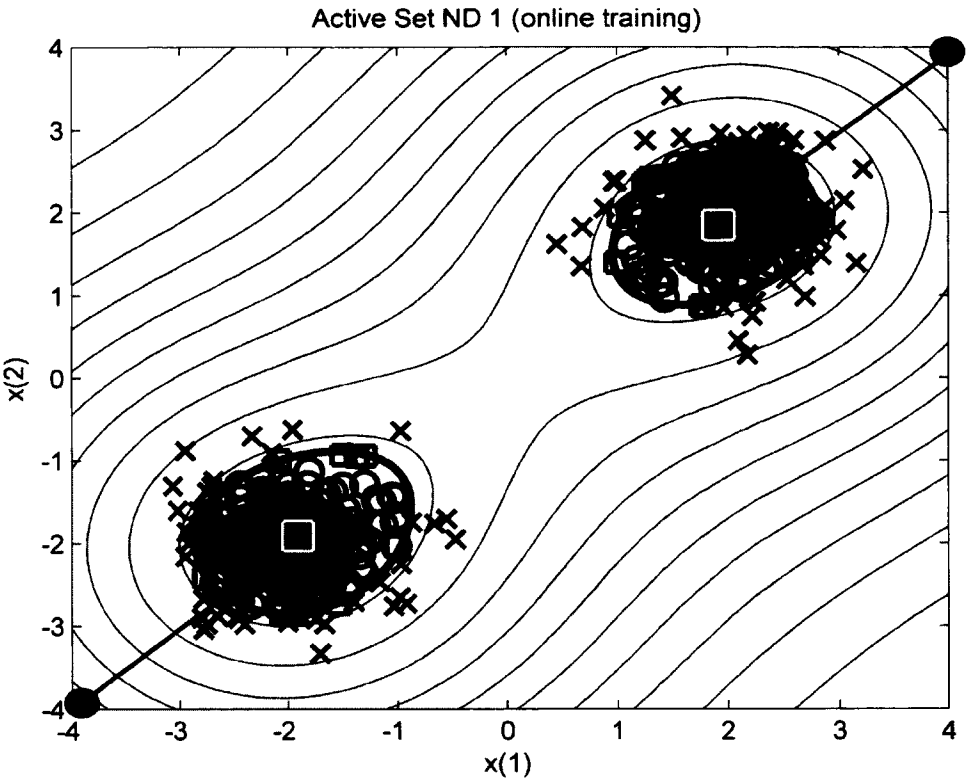


**Figure 49:** A modified version of the mean-shift algorithm was used to find the maximum of the resulting density estimate produced by the kernel machine (large square markers). Starting points before using the mean-shift are illustrated by the arrows in the plot.

on bound and non-bound support vectors. This will have the same computational benefit as the evaluation of the kernel expansion.

## 7.6 Conclusions and Application Scope

In this chapter an efficient training algorithm for a $\nu$-support vector novelty detector was presented. The active set training algorithm makes use of averaging across non-bound support vectors for calculation of the margin parameter $\rho$. The calculation of $\rho$ in this manner is suitable for online applications where the QP problem objective

function changes over time. The use of a similarity measure for ordering updates to $\alpha$ weights with respect to the current input allows the kernel machine to optimize the QP objective function, while at the same time optimize weights in a fashion that is directed by the input.

A modified mean shift procedure was presented and evaluated for the kernel expansion as defined by this active set algorithm. This computationally efficient modified mean shift procedure can be used for tracking the maximum in a given kernel density estimate, or for the detection of changes in the estimated probability density estimate.

The results of the AS algorithm provide both quantitative and qualitative evidence of its effectiveness. Qualitatively the results of the AS algorithm output and QP optimized kernel machine output look very similar in shape. The AS algorithm tends to produce more margin support vectors, but has a computational complexity lower than that of a generic QP optimizer. In the work of Schölkopf et al. [62] the authors provide bounds on the number of support vectors and outliers that their SMO algorithm produces. The results presented in this thesis do follow the bounds stated in [62], but the AS algorithm does not converge to the same exact solution as in [62] due to its online nature.

The novel contributions of the AS algorithm are outlined in Section 7.3.2. Additionally, the mode tracking approach presented in this chapter is novel contribution that follows Comaniciu's work [78, 80] for the tracking of the maximum of a kernel density estimator. Since the kernel machine enforces regularization in the QP solution the resulting mean shift procedure will only use the support vectors allowing for a more computationally efficient evaluation of the mean shift.

The application scope of the AS algorithm is in novelty detection and mode tracking. The training of an efficient kernel machine novelty detector can find use in computer vision, control systems, decision support systems and signal processing to name a few. Mode tracking can be directly used for the tracking of objects in video.

The advantage of using a kernel machine for mode tracking is that support vectors are used as a basis for the object being tracked, allowing for fewer expansion term evaluations when computing the mean shift.

# Chapter 8

# Concluding Remarks

Online environments give rise to four main problems in the effective use of kernel machines in intelligent systems. In Chapter 1 five separate algorithms were presented to deal with one or more of the following problem areas. From the research conducted in this thesis, conclusions can be made about the nature of each problem and the solution given. Below is a summary of all conclusions made for the problems addressed in this thesis.

- **Computational efficiency in kernel machine evaluation and training**: The primary conclusion regarding the efficiency of kernel machines in online environments is that the time taken to perform the kernel evaluations is the largest obstacle to efficient evaluation. Throughout this thesis the computational complexity was considered in terms of the training set size, or in terms of the number of expansion terms used for kernel machine evaluation. This thesis provides two contributions to the efficient execution or training of kernel machines.

  First, the S3 algorithm described in Chapter 3 gave a computationally efficient method of selecting a subset of support vectors for training and evaluation of a kernel machine classifier. The S3 algorithm can significantly reduce the number

of kernel expansion terms used for classification. The primary disadvantages of the S3 algorithm is that the accuracy of the classifier suffers in the presence of stationary data, and optimization of S3 parameters must be done offline.

Second, the AS algorithm presented in Chapter 7 provides a kernel machine that can be used for learning in online situations by solving the QP problem in an online fashion. The AS algorithm optimizes a QP problem that yields a result that is close to the traditionally QP trained $\nu$-support vector novelty detector. The AS algorithm is more computationally efficient than using a QP solver on a growing window of data. The disadvantage of the AS algorithm is that if the number of KKT violators is close to the training set size, the computational complexity will be close to $O(m^2)$.

Solving the problem of computational complexity is significant because if kernel machines are to be used in embedded or resource limited environments they must operate as efficiently as possible.

- **Adaptability of the kernel machine to the time-varying nature of the input data**: The results of this thesis show that online data sources can provide an infinite, time-varying data stream. The time-varying nature of the data is in direct contrast with the training method of offline kernel machines. If the data is time-varying, the kernel machine must be able to adapt the previously computed kernel weights and any required kernel parameters. There are five contributions to the adaptability of kernel machines to time-varying data presented in this thesis.

  First, the S3 algorithm of Chapter 3 selects support vectors based on their age and can therefore adapt to non-stationarity in the input stream. Second, the PKM algorithm of Chapter 4 uses a similarity measure to repeatedly update expansion weights using the KLMS update procedure. Third, the SDC algorithm

in Chapter 5 iteratively updates more expansion term weights randomly across the whole kernel machine buffer rather than just applying the KLMS algorithm to the newly arrived input. Fourth, the algorithm given in Chapter 6 shows how the RBF kernel parameter is updated based on kernel machine input statistics and fuzzy logic. Fifth, the AS algorithm makes use of a similarity measure with the current input when selecting KKT violation pairs for optimization. The disadvantage with these algorithms is the extra computational complexity that is involved with the adaptation procedure. The computation complexity was kept within $O(m)$ (except for the AS algorithm which is close to $O(m)$) which still makes the solutions suitable for online use. Extra computational overhead may not be appropriate in all situations.

The adaptation of kernel machines is significant because it maintains the accuracy of the kernel machine during online use on time-varying data. There are many online situations where adaptation is a system critical requirement such as in control systems and signal processing applications.

- **Accuracy of the resulting kernel machine in online situations**: The results presented in this thesis clearly show that it is possible to improve upon the classification or regression accuracy of both the NORMA and the KLMS algorithm in online applications. The use of stochastic gradient descent allows the kernel machine to take a step in minimizing the empirical risk with each new input. Both the NORMA and the KLMS algorithm operate with $O(m)$ computational complexity but only update the weight of the current input vector. A similarity measure between the current input and the stored support vectors allows for update of kernel machine weights and also prioritizes updates to weights that have the greatest effect on the current output. There are three contributions in this thesis when considering the accuracy of kernel machines.

First, the PKM algorithm presented in Chapter 4 uses a similarity measure for updating more than one expansion term weight per input time step. This represents a stochastic gradient descent of an empirical risk functional for the kernel machine. Second, the SDC-KLMS algorithm in Chapter 5 uses random selection method for updating more than one kernel expansion weight, and also uses an error correction heuristic when adding a new input to a finite sized buffer. Third, the AS algorithm in Chapter 7 achieves near the same level of accuracy as an offline QP solution for a kernel machine used for novelty detection. The contributions discussed are significant because solving the QP problem allows increased accuracy, but comes at a higher computational cost of between $O(m)$ and $O(m^2)$ [56].

- **The effect of limited memory environments on the accuracy of kernel machines**: The results presented in this thesis illustrate the effect of truncation error when using a finite size memory for kernel machines. The representer theorem states that the number of support vectors will increase without bound over time [40]. Therefore memory space will become a limited resource issue and older support vectors will have to be deleted in favor of new ones. The results described in Chapter 5 clearly illustrate the effect truncation error can have of the accuracy and stability of the KLMS kernel machine. The SDC-KLMS algorithm addresses the effect of truncation error [40] that occurs when a support vector is replaced in the memory buffer. The SDC-KLMS algorithm is significant because it addresses an issue with accuracy and also with stability for the use of kernel machines in online environments. The application scope of the SDC-KLMS algorithm is for finite memory applications. There are situations where memory is not an issue but the computational complexity of evaluating the kernel expansion is too high. In this case the buffer can be limited in

size to limit computational cost creating an issue with truncation error. The SDC-KLMS algorithm can be used in this setting as well.

The algorithms presented in this thesis will find use in various online pattern recognition applications. In the case of application setting where input data and memory are plentiful, the S3 algorithm can reduce computational complexity during training and expansion evaluation. When an online pattern recognition problem requires increased accuracy, and computational resources are available, then the PKM algorithm can be used. Applications with restricted memory may require that use of the SDC algorithm if truncation error is present. The kernel parameter adaptation algorithm and the AS quadratic programming algorithm are online algorithms and can be employed in an online setting to improve the performance of the kernel machine. Kernel parameter adaptation should be used when the nature of the input data can be partially estimated. The AS algorithm can be used for concept drift detection as well as other density estimation problems. All algorithms can be combined with each other in an online pattern recognition system using kernel machines. It is important to note that an independent evaluation of the effectiveness of combining the algorithms should be conducted in addition to the optimization of algorithm parameters.

There are many future research directions on the topic of specialized online kernel machines. As discussed above, one research area is the potential combination of the algorithms presented in this thesis. For example, both the S3 algorithm, and the PKM algorithm make use of the idea of subsets. It is possible to add a similarity measure to the S3 algorithm and use this metric instead of a metric based on age of the support vector in the buffer when composing the subset. As well it is possible to examine the idea of truncation error compensation with any of the other algorithms in this work.

Another interesting area of research is the optimization and integration of online kernel methods into hybrid intelligent systems. How the kernel machine is used within

the framework of an online hybrid intelligent system can have a large impact on the effectiveness of the kernel machine. As in the last decade with offline kernel machines, there are many opportunities for the development of kernel machines in the area of online intelligent systems.

# Appendix A

# Contributing Publications

Rhinelander, J., Liu, X.P., Stochastic Subset Selection for Learning With Kernel Machines., *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol.42, no.3, pp.616-626, June 2012.

Rhinelander, J., Liu, X.P., The partitioned kernel machine algorithm for online learning., *IEEE International Conference on Computational Intelligence for Measurement Systems and Applications (CIMSA) 2012*, vol., no., pp.1-6, 2-4 July 2012

Rhinelander, J., Liu, X.P. Adaptive kernels for data recovery in tele-haptic and tele-operation environments., *IEEE International Workshop on Haptic Audio Visual Environments and Games (HAVE) 2011*, pp.152-157, October 14-17, 2011.

Rhinelander, J., Liu, P.X., Tracking a moving hypothesis for visual data with explicit switch detection., *IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA) 2009*, vol., no., pp.1-8, 8-10 July 2009

Rhinelander, J., Liu, P.X., A Single-class Support Vector Machine Translation

Algorithm To Compensate For Non-stationary Data In Heterogeneous Vision-based Sensor Networks., *IEEE Instrumentation and Measurement Technology Conference Proceedings (IMTC) 2008*, pp.1102-1106, May 12-15 2008

# List of References


[1] B. Scholkpf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, 2002.

[2] J. Shaw-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis.* Cambridge University Press, 2004.

[3] V. N. Vapnik, *Statistical Learning Theory.* Wiley-Interscience, 1998.

[4] V. Vapnik, *The Nature of Statistical Learning Theory.* Springer, 2000.

[5] S. Abe, *Support Vector Machines for Pattern Classification.* Springer-Verlag London, 2005.

[6] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach, 2nd Ed.* Prentice Hall, Pearson Education Inc., 2003.

[7] J. C. Principe, N. R. Euliano, and W. C. Lefebvre, *Neural and Adaptive Systems: Fundamentals Through Simulations.* John Wiley & Sons Ltd., 2000.

[8] J. Zhou, D. Gao, and D. Zhang, "Moving vehicle detection for automatic traffic monitoring." *IEEE Transactions on Vehicular Technology*, vol. 56, no. 1, pp. 51–59, 2007.

[9] H. Sidenbladh, "Detecting human motion with support vector machines." in *Proceedings of the 17th International Conference on Pattern Recognition (ICPR04).* IEEE Computer Society, 2004.

[10] M. Harville and D. Li, "Fast, integrated person tracking and activity recognition with plan-view templates from a single stereo camera." in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR04).* IEEE Computer Society, 2004.

[11] H. Meng, N. Pears, and C. Bailey, "A human action recognition system for embedded computer vision application." in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR07).* IEEE, 2007, pp. 1–6.

[12] K. F. MacDorman, H. Nobuta, S. Koizumi, and H. Ishiguro, "Memory-based attention control for activity recognition at a subway station." *IEEE Multimedia*, vol. 14, no. 2, pp. 38–49, 2007.

[13] S. Maldonado-Bascn, S. Lafuente-Arroyo, P. Gil-Jimnez, H. Gmez-Moreno, and F. Lpez-Ferreras, "Road-sign detection and recognition based on support vector machines." *IEEE Transactions On Intelligent Transportation Systems*, vol. 8, no. 2, pp. 264–278, 2007.

[14] K. C. Yang, C. C. Guest, and P. Das, "Motion blur detecting by support vector machine." *Mathematical Methods in Pattern and Image Analysis*, vol. 5916, 2005.

[15] S. Moustakidis, J. Theocharis, and G. Giakas, "Subject recognition based on ground reaction force measurements of gait signals." *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 38, no. 6, pp. 1476 –1485, December 2008.

[16] J. Zhu, S. Hoi, and M.-T. Lyu, "Robust regularized kernel regression." *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 38, no. 6, pp. 1639 –1644, December 2008.

[17] L. Zhang, W. Zhou, and L. Jiao, "Wavelet support vector machine." *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 34, no. 1, pp. 34–39, February 2004.

[18] Y.-H. Liu, H.-P. Huang, and C.-H. Weng, "Recognition of electromyographic signals using cascaded kernel learning machine." *IEEE/ASME Transactions on Mechatronics*, vol. 12, no. 3, pp. 253 –264, June 2007.

[19] G. Wang, Y. Li, and D. Bi, "Support vector machine networks for friction modeling." *IEEE/ASME Transactions on Mechatronics*, vol. 9, no. 3, pp. 601 –606, September 2004.

[20] Y. Li and D. Bi, "A method for dynamics identification for haptic display of the operating feel in virtual environments." *IEEE/ASME Transactions on Mechatronics*, vol. 8, no. 4, pp. 476 –482, December 2003.

[21] X. Yuan, Y. Wang, and L. Wu, "Svm-based approximate model control for electronic throttle valve." *IEEE Transactions on Vehicular Technology*, vol. 57, no. 5, pp. 2747–2756, 2008.

[22] L. Wang, Z. Liu, C. Chen, Y. Zhang, S. Lee, and X. Chen, "Energy-efficient svm learning control system for biped walking robots." *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 5, pp. 831–837, 2013.

[23] G. Liu, L. Chen, W. Zhao, Y. Jiang, and L. Qu, "Internal model control of permanent magnet synchronous motor using support vector machine generalized inverse." *IEEE Transactions on Industrial Informatics*, vol. 9, no. 2, pp. 890–898, 2013.

[24] P.-K. Wong, Q. Xu, C.-M. Vong, and H.-C. Wong, "Rate-dependent hysteresis modeling and control of a piezostage using online support vector machine and relevance vector machine." *IEEE Transactions on Industrial Electronics*, vol. 59, no. 4, pp. 1988–2001, 2012.

[25] M. Oskoei and H. Hu, "Support vector machine-based classification scheme for myoelectric control applied to upper limb." *IEEE Transactions on Biomedical Engineering*, vol. 55, no. 8, pp. 1956–1965, 2008.

[26] Z. Li, J. Zhang, and Y. Yang, "Motion control of mobile under-actuated manipulators by implicit function using support vector machines." *IET Control Theory Applications*, vol. 4, no. 11, pp. 2356–2368, 2010.

[27] B. Olson, J. Si, J. Hu, and J. He, "Closed-loop cortical control of direction using support vector machines." *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 13, no. 1, pp. 72–80, 2005.

[28] Z. Haoran, H. Zhengzhi, F. Rui, and Y. Zhiqiang, "Support vector machine-based nonlinear system modeling and control." *Journal of Systems Engineering and Electronics*, vol. 14, no. 3, pp. 53–58, 2003.

[29] C. Gaudes, I. Santamaria, J. Via, E. Gomez, and T. Paules, "Robust array beamforming with sidelobe control using support vector machines." *IEEE Transactions on Signal Processing*, vol. 55, no. 2, pp. 574–584, 2007.

[30] L. Jian, C. Gao, and Z. Xia, "Constructing multiple kernel learning framework for blast furnace automation." *IEEE Transactions on Automation Science and Engineering*, vol. 9, no. 4, pp. 763–777, 2012.

[31] F. Bellocchio, N. Borghese, S. Ferrari, and V. Piuri, "Kernel regression in hrbf networks for surface reconstruction." in *IEEE International Workshop on Haptic Audio visual Environments and Games (HAVE), 2008.*, October 2008, pp. 160 –165.

[32] D. Ming, C. Zhang, Y. Bai, B. Wan, Y. Hu, and K. Luk, "Gait recognition based on multiple views fusion of wavelet descriptor and human skeleton model." in *IEEE International Conference on Virtual Environments, Human-Computer Interfaces and Measurements Systems, 2009. VECIMS '09.*, May 2009, pp. 246 –249.

[33] B. Wan, Y. Liu, D. Ming, H. Qi, Y. Wang, and R. Zhang, "Feature recognition of multi-class imaginary movements in brain-computer interface." in *IEEE International Conference on Virtual Environments, Human-Computer Interfaces and Measurements Systems, 2009. VECIMS '09.*, May 2009, pp. 250 –254.

[34] G. Wang, "A survey on training algorithms for support vector machine classifiers." in *Networked Computing and Advanced Information Management, 2008. NCM '08. Fourth International Conference on*, vol. 1, 2008, pp. 123–128.

[35] N. Sapankevych and R. Sankar, "Time series prediction using support vector machines: A survey." *IEEE Computational Intelligence Magazine*, vol. 4, no. 2, pp. 24 –38, May 2009.

[36] N. Murata, M. Kawanabe, A. Ziehe, K. Muller, and S. Amari, "On-line learning in changing environments with applications in supervised and unsupervised learning." *Neural Networks*, vol. 15, pp. 743–760, 2002.

[37] D. Sebald and J. Bucklew, "Support vector machine techniques for nonlinear equalization." *IEEE Transactions on Signal Processing*, vol. 48, no. 11, pp. 3217 –3226, Nov 2000.

[38] G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning." in *Advances in Neural Information Processing Systems (NIPS)*, 2000, pp. 409–415.

[39] L. Csató and M. Opper, "Sparse representation for gaussian process models." in *Advances in Neural Information Processing Systems (NIPS)*, 2000, pp. 444–450.

[40] J. Kivinen, A. J. Smola, and R. C. Williamson, "Online learning with kernels." *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2165–2176, 2004.

[41] M. Awad, X. Jiang, and Y. Motai, "Incremental support vector machine framework for visual sensor networks." *EURASIP Journal on Advances in Signal Processing*, vol. 2007, pp. 1–15, 2007.

[42] M. Davy, F. Desobry, A. Gretton, and C. Doncarli, "An online support vector machine for abnormal events detection." *Signal Processing*, vol. 86, no. 8, pp. 2009–2025, 2006.

[43] F. Rosenblatt, *Principles of Neurodynamics.* Spartan Books, 1969.

[44] T. Graepel, R. Herbrich, and R. C. Williamson, "From margin to sparsity." *Advances in Neural Information Processing Systems*, vol. 13, pp. 210–216, 2001.

[45] K. Crammer, J. Kandola, and Y. Singer, "Online classification on a budget." *Advances in Neural Information Processing Systems*, vol. 16, 2003.

[46] O. Dekel, S. Shalev-Shwartz, and Y. Singer, "The forgetron: A kernel-based perceptron on a budget." *SIAM Journal on Computing*, vol. 37, no. 5, pp. 1342–1372, 2008.

[47] T.-T. Frie, N. Cristianini, and C. Campbell, "The kernel-adatron algorithm: a fast and simple learning procedure for support vector machines." in *Machine Learning: Proceedings of the Fifteenth International Conference.* Morgan Kaufmann Publishers, 1998.

[48] N. Cesa-Bianchi and C. Gentile, "Tracking the best hyperplane with a simple budget perceptron." *Lecture Notes in Computer Science*, vol. 4005, 2006.

[49] Y. Li and P. M. Long, "The relaxed online maximum margin algorithm." *Machine Learning*, vol. 46, pp. 361–387, 2002.

[50] C. Gentile, "A new approximate maximal margin classification algorithm." *Journal of Machine Learning Research*, vol. 2, pp. 213–242, 2001.

[51] W. Liu, J. Principe, and S. Haykin, *Kernel Adaptive Filtering.* John Wiley & Sons, Inc, 2010.

[52] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive Least-Squares algorithm." *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2275–2285, 2004.

[53] W. Liu, P. Pokharel, and J. Principe, "The kernel least-mean-square algorithm." *IEEE Transactions on Signal Processing*, vol. 56, no. 2, pp. 543–554, 2008.

[54] G. Kimeldorf and G. Wahba, "Some results on tchebycheffian spline functions." *Journal of Mathematical Analysis and Applications*, vol. 33, no. 1, pp. 82 – 95, 1971.

[55] S. Vishwanathan, N. N. Schraudolph, and A. J. Smola, "Step size adaptation in reproducing kernel hilbert space." *Journal of Machine Learning Research*, vol. 7, pp. 1107–1133, 2006.

[56] J. C. Platt, "Fast training of support vector machines using sequential minimal optimization." *Advances in kernel methods: support vector learning*, pp. 185–208, 1999.

[57] R. Klinkenberg and T. Joachims, "Detecting concept drift with support vector machines." in *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 487–494.

[58] J. Rhinelander and X. Liu, "Stochastic subset selection for learning with kernel machines." *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 42, no. 3, pp. 616–626, June 2012.

[59] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition." *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[60] W. N. Street and Y. Kim, "A streaming ensemble algorithm (sea) for large-scale classification." in *Proceedings of The Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '01, 2001, pp. 377–382.

[61] J. Rhinelander and X. Liu, "The partitioned kernel machine algorithm for online learning." in *IEEE International Conference on Computational Intelligence for Measurement Systems and Applications (CIMSA), 2012*, July 2012, pp. 1 –6.

[62] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution." *Neural Computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[63] A. Ruszczynski, *Nonlinear Optimization.* Princeton University Press, 2006.

[64] C. Cortes and V. Vapnik, "Support vector networks." *Machine Learning*, vol. 20, pp. 273–297, 1995.

[65] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: An ensemble method for drifting concepts." *Journal Of Machine Learning Research*, vol. 8, pp. 2755–2790, December 2007.

[66] Z.-H. Zhou, J. Wu, and W. Tang, "Ensembling neural networks: Many could be better than all." *Artificial Intelligence*, vol. 137, no. 1-2, pp. 239 - 263, 2002.

[67] M. Karnick, M. Muhlbaier, and R. Polikar, "Incremental learning in non-stationary environments with concept drift using a multiple classifier based approach." in *19th International Conference on Pattern Recognition (ICPR), 2008.*, December 2008, pp. 1 –4.

[68] R. Elwell and R. Polikar, "Incremental learning in nonstationary environments with controlled forgetting." in *International Joint Conference on Neural Networks, 2009. IJCNN 2009.*, June 2009, pp. 771 –778.

[69] T. M. Mitchell, R. Caruana, D. Freitag, J. McDermott, and D. Zabowski, "Experience with a learning personal assistant." *Communications of the ACM*, vol. 37, no. 7, pp. 80–91, 1994.

[70] R. Klinkenberg and I. Renz, "Adaptive information filtering: Learning in the presence of concept drifts." in *Workshop Notes of the ICML/AAAI-98 Workshop Learning for Text Categorization.* AAAI Press, 1998, pp. 33–40.

[71] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts." *Machine Learning*, vol. 23, no. 1, pp. 69–101, Apr. 1996.

[72] J. Rhinelander and P. Liu, "Tracking a moving hypothesis for visual data with explicit switch detection." in *IEEE Symposium on Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009.*, July 2009, pp. 1–8.

[73] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory." in *Proceedings of the Sixth International Symposium on Micro Machine and Human Science (MHS), 1995*, oct 1995, pp. 39 –43.

[74] J. Rhinelander and X. Liu, "Adaptive kernels for data recovery in tele-haptic and tele-operation environments." in *IEEE International Workshop on Haptic Audio Visual Environments and Games (HAVE).*, October 2011.

[75] K. Ucak and G. Oke, "Adaptive pid controller based on online lssvr with kernel tuning." in *2011 International Symposium on Innovations in Intelligent Systems and Applications (INISTA)*, June 2011, pp. 241–247.

[76] J. Sun, "Fast tuning of svm kernel parameter using distance between two classes." in *3rd International Conference on Intelligent System and Knowledge Engineering (ISKE), 2008.*, vol. 1, November 2008, pp. 108–113.

[77] A. Singh and J. Principe, "Kernel width adaptation in information theoretic cost functions." in *2010 IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP)*, march 2010, pp. 2062 –2065.

[78] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 5, pp. 564 – 577, May 2003.

[79] E. Osuna, R. Freund, and F. Girosi, "An improved training algorithm for support vector machines." in *Proceedings of the 1997 IEEE Workshop Neural Networks for Signal Processing [1997] VII.*, Sep 1997, pp. 276–285.

[80] D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, May 2002.