Adaptive Support Vector Machines for Regression

M. Palaniswami and A. Shilton

Department of Electrical and Electronic Engineering
The University of Melbourne, Victoria-3010, Australia
apsh@ee.mu.oz.au, swami@ee.mu.oz.au

ABSTRACT

Support Vector Machines are a general formulation for machine learning. It has been shown to perform extremely well for a number of problems in classification and regression. However, in many difficult problems, the system dynamics may change with time and the resulting new information arriving incrementally will provide additional data. At present, there is limited work to cope with the computational demands of modeling time varying systems. Therefore, we develop the concept of adaptive support vector machines that can learn from incremental data. In this paper, results are provided to demonstrate the applicability of the adaptive support vector machines techniques for pattern classification and regression problems.

1. INTRODUCTION

Support Vector Machines are a new class of learning machines. A learning machine here is represented by a family of functions f($\mathbf{x}$, $\boldsymbol{\alpha}$) where $\mathbf{x}$ represents input and $\boldsymbol{\alpha}$ represents adjustable parameters. Training the learning machine involves appropriate fixing of the adjustable parameters by minimizing the training error and a risk term that is related to the capacity of the learning machine. Conceptually, SVMs work in a higher dimensional space called the feature space [Vapnik, 1995, 1998; Burges, 1998]. The feature space is related to the input data space through a transformation. This transformation can, in general, be nonlinear. In the context of pattern classification, the classification is achieved by specifying hyperplanes in the feature space separating the classes. Thus the nonlinear boundary in the input space that separates the classes is transformed to a hyperplane in the feature space. The vectors that lie close to these hyperplanes in the feature space are called the support vectors. The automatic identification of these vectors and the calculation of the hyperplanes through the SVM formulation lead to a standard optimization problem known as the convex quadratic programming problem [Fletcher, 1987; Murty, 1988].

It is well known that many signal processing and time series problems are essentially processing long data records with large data sets [8]. Many of these problems can also be charaterized by non-stationarity and noise. When new data arrives, the problem is effectively solved from the beginning and this is extremely time consuming. Therefore, there is a need to develop incremental learning techniques for the SVMs to adapt to changing requirements defining the problem. In our earlier paper, we have given incremental SVM results for pattern classification problems that deal with sequentially arriving video images [9, 10, 11]. In this paper, we extend our earlier work [9,10] to cover regression problems in the same setting and demonstrate the feasibility of such methods. There exist applications for such techniques in fields ranging from time-series prediction, function approximation and adaptive control, and various other related applications.

2. SUPPORT VECTOR MACHINES

In this section, we give a brief introduction to the theory of support vector machine learning for pattern recognition and regression. We then give a general formulation of the problem, and demonstrate that both pattern recognition and regression are special cases of this form.

2.1. Pattern Recognition

We now look briefly at how the SVM pattern recognition problem is formulated [4]. Suppose we are given the training set:

$$\begin{aligned} &\underset{P}{\Theta} = \{(\mathbf{x}_1, d_1), (\mathbf{x}_2, d_2), \ldots, (\mathbf{x}_N, d_N)\} \\ &\mathbf{x}_i \in \Re^{d_L} \\ &d_i \in \{+1, -1\} \end{aligned} \tag{2.1}$$

It is assumed that this training set was produced by some unknown but well defined map:

$$\begin{aligned} &\hat{g}: \Re^{d_L} \to \{+1, -1\} \\ &\text{so that:} \quad \hat{d}_i = \hat{g}\left(\hat{\mathbf{x}}_i\right) \\ &\qquad\quad \mathbf{x}_i = \hat{\mathbf{x}}_i + \text{noise} \end{aligned} \tag{2.2}$$

We also (implicitly, as shown later) define a mapping from our input space to a (usually) higher dimensional feature space as:

$$\boldsymbol{\varphi}(\mathbf{x}) = \begin{bmatrix} \varphi_1(\mathbf{x}) \\ \varphi_2(\mathbf{x}) \\ \vdots \\ \varphi_{d_H}(\mathbf{x}) \end{bmatrix} \in \Re^{d_H} \tag{2.3}$$

Let us suppose that our two training classes are linearly separable when mapped into feature space. Then we can define a discriminant function *g(***x***)* thus:

$$\begin{aligned} &g(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}) + b \\ &g(\mathbf{x}_i) > 0 \quad \forall \quad d_i = +1 \\ &g(\mathbf{x}_i) < 0 \quad \forall \quad d_i = -1 \end{aligned} \tag{2.4}$$

The discriminant function may be used to classify points of unknown classification as follows:

$$d_{\mathbf{y}} = \text{sgn}\left[g(\mathbf{y})\right] \tag{2.5}$$

Any discriminant function (2.4) defines a linear (flat) decision surface in feature space that bisects our two classes. This plane is characterised by:

$$\mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}) + b = 0 \tag{2.6}$$

However, there will be infinitely many such planes. To select the surface best suited to the task, the SVM maximises the distance between the decision surface and those training points lying closest to it (the support vectors). It is easy to show (see [2] for example) that this distance is inversely proportional to $\frac{1}{2}\mathbf{w}^T\mathbf{w}$.

Furthermore, our training data may not be linearly separable in feature space. To account for this, we use slack variables to allow some points to lie on the wrong side of the decision surface, but include a term in our problem to penalise such incorrect classifications.

Using a linear penalty function, this leads to the problem [2]:

$$\begin{aligned} &\min_{\mathbf{w},b,\boldsymbol{\xi}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\mathbf{t}^T\boldsymbol{\xi} \\ &\text{such that} \quad d_i\left(\mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) + b\right) \geq s_i - \xi_i \\ &\qquad\quad \boldsymbol{\xi} \geq \mathbf{0} \end{aligned} \tag{2.7}$$

Where $s_i, t_i > 0$ are user constants, both usually set to 1. s may be used to set the relative distance of a training point from the decision surface, and t to penalise the incorrect classification of some points more strongly than others.

Unfortunately, (2.7) presents computationally difficulties (especially if d_H is large), so we instead use the (primal-) dual form [2][1]:

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \max_{b} \quad \tfrac{1}{2}\begin{bmatrix}\boldsymbol{\alpha}\\ b\end{bmatrix}^T \begin{bmatrix}\mathbf{G} & \mathbf{d}\\ \mathbf{d}^T & 0\end{bmatrix}\begin{bmatrix}\boldsymbol{\alpha}\\ b\end{bmatrix} - \begin{bmatrix}\boldsymbol{\alpha}\\ b\end{bmatrix}^T \begin{bmatrix}\mathbf{s}\\ 0\end{bmatrix} \\ &\text{such that} \quad \mathbf{0} \leq \boldsymbol{\alpha} \leq C\mathbf{t} \end{aligned} \tag{2.8}$$

Where:

$$\begin{aligned} G &\in \Re^{N \times N} \\ G_{ij} &= d_i d_j K(\mathbf{x}_i, \mathbf{x}_j) \\ K(\mathbf{x}_i, \mathbf{x}_j) &= \boldsymbol{\varphi}(\mathbf{x}_i)^T \boldsymbol{\varphi}(\mathbf{x}_j) \\ &= \text{kernel function} \end{aligned} \tag{2.9}$$

We can express our discriminant as:

$$g(\mathbf{y}) = \sum_{i=1}^{N} \alpha_i d_i K(\mathbf{x}_i, \mathbf{y}) + b \tag{2.10}$$

We will be using the (primal-) dual form (2.8) in this paper. Note that:

- There is exactly one α for every training vector.
- Only those α's corresponding to support vectors will be non-zero. Hence the SVM can be fully specified using only the support vectors. Furthermore, the support vectors usually make up only a small fraction of all training vectors.
- The matrix $\mathbf{G}$ is positive semi-definite and the constraints are linear. Hence the program is convex. This implies that any local optimum will also be a global (although not necessarily unique) optimum.
- The dimension of the feature space is hidden by the kernel function $K(\mathbf{x}, \mathbf{z})$, circumventing the "curse of dimensionality" which affects the primal form.
- So long as our kernel function satisfies Mercer's condition [1], we need never explicitly know what our feature mapping actually is.

[1] Usually, b is not included in the dual. However, this results in an additional equality constraint, which leads to computational difficulties.

We now look at the optimality conditions for (2.8). Define:

$$\begin{bmatrix} \mathbf{e} \\ f \end{bmatrix} = \begin{bmatrix} \mathbf{G} & \mathbf{d} \\ \mathbf{d}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ b \end{bmatrix} - \begin{bmatrix} \mathbf{s} \\ 0 \end{bmatrix} \tag{2.11}$$

The KKT conditions for (2.8) can then be written:

$$\begin{aligned} &\mathbf{0} \le \boldsymbol{\alpha} \le C\mathbf{t} \\ &f = 0 \\ &e_i \begin{cases} \ge 0 & \text{if } \alpha_i = 0 \\ \le 0 & \text{if } \alpha_i = Ct_i \\ = 0 & \text{if } 0 < \alpha_i < Ct_i \end{cases} \end{aligned} \tag{2.12}$$

2.2. Regression

We now look briefly at how the SVM regression problem is formulated [7]. Suppose we are given the training set:

$$\begin{aligned} &\underset{R_=}{\Theta} = \left\{ (\mathbf{x}_1, z_1), (\mathbf{x}_2, z_2), \ldots, (\mathbf{x}_{N_=}, z_{N_=}) \right\} \\ &\mathbf{x}_i \in \Re^{d_L} \\ &z_i \in \Re \end{aligned} \tag{2.13}$$

Once again, it is assumed that this training set was produced by some unknown but well defined map:

$$\begin{aligned} &\hat{g} : \Re^{d_L} \to \Re \\ &\text{so that:} \quad z_i = \hat{g}(\hat{\mathbf{x}}_i) + \text{noise} \\ &\qquad\qquad \mathbf{x}_i = \hat{\mathbf{x}}_i + \text{noise} \end{aligned} \tag{2.14}$$

As for pattern recognition, we (implicitly) define a mapping from input to feature space as:

$$\boldsymbol{\varphi}(\mathbf{x}) = \begin{bmatrix} \varphi_1(\mathbf{x}) \\ \varphi_2(\mathbf{x}) \\ \vdots \\ \varphi_{d_H}(\mathbf{x}) \end{bmatrix} \in \Re^{d_H} \tag{2.15}$$

Using the set of functions $\varphi_i : \Re^{d_L} \to \Re$, we define a nonlinear approximation to $\hat{g}$:

$$g(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}) + b \tag{2.16}$$

Which may be interpreted as a linear function of position in feature space. Usually, support vector machines select $\mathbf{w}$ and b by minimising the functional:

$$\begin{aligned} &R(\mathbf{w}, b) = \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\mathbf{t}^T\boldsymbol{\xi} + C\mathbf{t}^T\boldsymbol{\xi}^* \\ &\text{where:} \quad \left(\mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) + b\right) \ge (z_i - \varepsilon_i) - \xi_i \\ &\qquad -\left(\mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) + b\right) \ge (-z_i - \varepsilon_i) - \xi_i^* \\ &\qquad \boldsymbol{\xi}, \boldsymbol{\xi}^* \ge \mathbf{0} \end{aligned} \tag{2.17}$$

where $\varepsilon_i > 0$ is a user constant, usually the same for all training points, that defines the training error allowed for a given point before the error is penalised (imparting a degree of noise insensitivity). The second element of (2.17) is a simple measure of empirical risk, whilst the first element is a regularisation term intended to prevent over-fitting.

This is computationally difficult. Therefore we use the (primal-) dual form [7]:

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \max_{b} \quad \tfrac{1}{2}\begin{bmatrix} \boldsymbol{\alpha} \\ b \end{bmatrix}^T \begin{bmatrix} \mathbf{G} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ b \end{bmatrix} - \begin{bmatrix} \boldsymbol{\alpha} \\ b \end{bmatrix}^T \begin{bmatrix} \mathbf{z} \\ 0 \end{bmatrix} + \begin{bmatrix} |\boldsymbol{\alpha}| \\ b \end{bmatrix}^T \begin{bmatrix} \boldsymbol{\varepsilon} \\ 0 \end{bmatrix} \\ &\text{such that} \quad -C\mathbf{t} \le \boldsymbol{\alpha} \le C\mathbf{t} \end{aligned} \tag{2.18}$$

Where:

$$\begin{aligned} &|\boldsymbol{\alpha}| = \begin{bmatrix} |\alpha_1| & |\alpha_2| & \ldots & |\alpha_N| \end{bmatrix}^T \\ &G \in \Re^{N \times N} \\ &G_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) \\ &K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\varphi}(\mathbf{x}_i)^T \boldsymbol{\varphi}(\mathbf{x}_j) \\ &\qquad = \text{kernel function} \end{aligned} \tag{2.19}$$

The resultant approximation function may be written:

$$g(\mathbf{y}) = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}_i, \mathbf{y}) + b \tag{2.20}$$

The properties of the SVM formulation are essentially the same as those listed for the pattern recognition case. Rather than repeating them here, we will move straight to the optimality conditions for the regression problem (2.18). Define:

$$\begin{bmatrix} \mathbf{e} \\ f \end{bmatrix} = \begin{bmatrix} \mathbf{G} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ b \end{bmatrix} - \begin{bmatrix} \mathbf{z} \\ 0 \end{bmatrix} \tag{2.21}$$

The KKT conditions for (2.18) can then be written:

$$
\begin{aligned}
&-C\mathbf{t} \le \boldsymbol{\alpha} \le C\mathbf{t} \\
&f = 0 \\
&e_i \begin{cases} \in [-\varepsilon_i, \varepsilon_i] & \text{if } \alpha_i = 0 \\ \le -\varepsilon_i & \text{if } \alpha_i = Ct_i \\ \ge \varepsilon_i & \text{if } \alpha_i = -Ct_i \\ = -\varepsilon_i & \text{if } 0 < \alpha_i < Ct_i \\ = \varepsilon_i & \text{if } -Ct_i < \alpha_i < 0 \end{cases}
\end{aligned} \tag{2.22}
$$

2.2. Generic SVMs - Regression with Inequalities

We now show how the SVM regression model may be extended to include inequalities. Suppose that, in addition to (2.13), we are given the data:

$$
\begin{aligned}
&\underset{R_{\ge}}{\Theta} = \left\{ (\mathbf{x}_1, z_1), (\mathbf{x}_2, z_2), \ldots, (\mathbf{x}_{N_{\ge}}, z_{N_{\ge}}) \right\} \\
&\underset{R_{\le}}{\Theta} = \left\{ (\mathbf{x}_1, z_1), (\mathbf{x}_2, z_2), \ldots, (\mathbf{x}_{N_{\le}}, z_{N_{\le}}) \right\} \\
&\mathbf{x}_i \in \Re^{d_L} \\
&z_i \in \Re
\end{aligned} \tag{2.23}
$$

During training, we want (if possible) to enforce the bounds:

$$
\begin{aligned}
&g(\mathbf{x}_i) \ge z_i \quad \forall \quad (\mathbf{x}_i, z_i) \in \underset{R_{\ge}}{\Theta} \\
&g(\mathbf{x}_i) \le z_i \quad \forall \quad (\mathbf{x}_i, z_i) \in \underset{R_{\le}}{\Theta}
\end{aligned} \tag{2.24}
$$

We do this by extending the functional (2.17) as:

$$
\begin{aligned}
&R(\mathbf{w}, b) = \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\mathbf{t}^T\boldsymbol{\xi} + C\mathbf{t}^T\boldsymbol{\xi}^* + C\mathbf{t}^T\boldsymbol{\xi}^{\ge} + C\mathbf{t}^T\boldsymbol{\xi}^{*\le} \\
&\text{where: } \left(\mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) + b\right) \ge (z_i - \varepsilon_i) - \xi_i \quad \forall \quad (\mathbf{x}_i, z_i) \in \underset{R_{=}}{\Theta} \\
&\quad -\left(\mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) + b\right) \ge (-z_i - \varepsilon_i) - \xi_i^* \quad \forall \quad (\mathbf{x}_i, z_i) \in \underset{R_{=}}{\Theta} \\
&\quad \left(\mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) + b\right) \ge z_i - \xi_i^{\ge} \quad \forall \quad (\mathbf{x}_i, z_i) \in \underset{R_{\ge}}{\Theta} \\
&\quad -\left(\mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) + b\right) \ge -z_i - \xi_i^{*\le} \quad \forall \quad (\mathbf{x}_i, z_i) \in \underset{R_{\le}}{\Theta} \\
&\quad \boldsymbol{\xi}^{\ge}, \boldsymbol{\xi}^{*\le}, \boldsymbol{\xi}, \boldsymbol{\xi}^* \ge \mathbf{0}
\end{aligned} \tag{2.25}
$$

The (primal-) dual optimisation problem (2.18) becomes:

$$
\begin{aligned}
&\min_{\boldsymbol{\alpha}} \max_{b} \quad \tfrac{1}{2}\begin{bmatrix} \boldsymbol{\alpha} \\ b \end{bmatrix}^T \begin{bmatrix} \mathbf{G} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ b \end{bmatrix} - \begin{bmatrix} \boldsymbol{\alpha} \\ b \end{bmatrix}^T \begin{bmatrix} \mathbf{z} \\ 0 \end{bmatrix} + \begin{bmatrix} |\boldsymbol{\alpha}| \\ b \end{bmatrix}^T \begin{bmatrix} \boldsymbol{\varepsilon} \\ 0 \end{bmatrix} \\
&\text{such that} \quad -Ct_i \le \alpha_i \le Ct_i \quad \forall \quad (\mathbf{x}_i, z_i) \in \underset{R_{=}}{\Theta} \\
&\qquad\qquad 0 \le \alpha_i \le Ct_i \quad \forall \quad (\mathbf{x}_i, z_i) \in \underset{R_{\ge}}{\Theta} \\
&\qquad\qquad -Ct_i \le \alpha_i \le 0 \quad \forall \quad (\mathbf{x}_i, z_i) \in \underset{R_{\le}}{\Theta}
\end{aligned} \tag{2.26}
$$

Which satisfies the usual advantageous properties of SVM formulations. As was done previously, we define:

$$
\begin{bmatrix} \mathbf{e} \\ f \end{bmatrix} = \begin{bmatrix} \mathbf{G} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ b \end{bmatrix} - \begin{bmatrix} \mathbf{z} \\ 0 \end{bmatrix} \tag{2.27}
$$

The KKT conditions for (2.26) can then be written:

$$
\begin{aligned}
&-Ct_i \le \alpha_i \le Ct_i \text{ if } (\mathbf{x}_i, z_i) \in \underset{R_{=}}{\Theta} \\
&0 \le \alpha_i \le Ct_i \text{ if } (\mathbf{x}_i, z_i) \in \underset{R_{\ge}}{\Theta} \\
&-Ct_i \le \alpha_i \le 0 \text{ if } (\mathbf{x}_i, z_i) \in \underset{R_{\le}}{\Theta} \\
&f = 0 \\
&e_i \begin{cases} \in [-\varepsilon_i, \varepsilon_i] & \text{if } \alpha_i = 0 \text{ and } (\mathbf{x}_i, z_i) \in \underset{R_{=}}{\Theta} \\ \ge 0 & \text{if } \left(\alpha_i = 0 \text{ and } (\mathbf{x}_i, z_i) \in \underset{R_{\ge}}{\Theta}\right) \\ & \text{or } \left(\text{if } \alpha_i = -Ct_i \text{ and } (\mathbf{x}_i, z_i) \in \underset{R_{\le}}{\Theta}\right) \\ \le 0 & \text{if } \left(\alpha_i = 0 \text{ and } (\mathbf{x}_i, z_i) \in \underset{R_{\le}}{\Theta}\right) \\ & \text{or } \left(\alpha_i = Ct_i \text{ and } (\mathbf{x}_i, z_i) \in \underset{R_{\ge}}{\Theta}\right) \\ \le -\varepsilon_i & \text{if } \alpha_i = Ct_i \text{ and } (\mathbf{x}_i, z_i) \in \underset{R_{=}}{\Theta} \\ \ge \varepsilon_i & \text{if } \alpha_i = -Ct_i \text{ and } (\mathbf{x}_i, z_i) \in \underset{R_{=}}{\Theta} \\ = -\varepsilon_i & \text{if } 0 < \alpha_i < Ct_i \text{ and } (\mathbf{x}_i, z_i) \in \underset{R_{=}}{\Theta} \\ = \varepsilon_i & \text{if } -Ct_i < \alpha_i < 0 \text{ and } (\mathbf{x}_i, z_i) \in \underset{R_{=}}{\Theta} \\ = 0 & \text{if } \left(0 < \alpha_i < Ct_i \text{ and } (\mathbf{x}_i, z_i) \in \underset{R_{\ge}}{\Theta}\right) \\ & \text{or } \left(\text{if } -Ct_i < \alpha_i < 0 \text{ and } (\mathbf{x}_i, z_i) \in \underset{R_{\le}}{\Theta}\right) \end{cases}
\end{aligned} \tag{2.28}
$$

Note that the SVM pattern recognition problem may be formulated as regression in inequalities. We form our regression training set from (2.1) as follows:

$$
\begin{aligned}
&\underset{R_{=}}{\Theta} = \varnothing \\
&\underset{R_{\ge}}{\Theta} = \left\{ (\mathbf{x}_i, s_i) : (\mathbf{x}_i, d_i) \in \underset{P}{\Theta}, d_i = +1 \right\} \\
&\underset{R_{\le}}{\Theta} = \left\{ (\mathbf{x}_i, -s_i) : (\mathbf{x}_i, d_i) \in \underset{P}{\Theta}, d_i = -1 \right\}
\end{aligned} \tag{2.29}
$$

Hence we need only consider the SVM training problem of form (2.26) and the associated optimality conditions (2.27) and (2.28). Pattern recognition and standard regression may be treated as special cases of the more general regression in inequalities problem.

3. OPTIMISING SVMs – THE ACTIVE SET APPROACH

As noted earlier, a feature of SVMs is that, for each training pair we have exactly one dual variable, α. The support vectors are those for which this variable is non-zero. It has been noted by a number of authors that support vectors typically make up only a small fraction of all training vectors.

Active set methods [3] allow us to take full advantage of this feature of the problem by reducing the effective dimensionality of the problem ((2.26) in this case) from the number of training points to the number of support vectors, as we will now describe.

3.1. Partitioning the Problem

Given a vector $\boldsymbol{\alpha}$, we partition it as follows:

$$\boldsymbol{\alpha} = \begin{bmatrix} \boldsymbol{\alpha}_{A1} \in \Re^{N_{A1}} = \mathbf{0} \\ \boldsymbol{\alpha}_{A2+} \in \Re^{N_{A2+}} = C\mathbf{t} \\ \boldsymbol{\alpha}_{A2-} \in \Re^{N_{A2-}} = -C\mathbf{t} \\ \boldsymbol{\alpha}_{B+} \in \Re^{N_{B+}} \in [\mathbf{0}, C\mathbf{t}] \\ \boldsymbol{\alpha}_{B-} \in \Re^{N_{B+}} \in [-C\mathbf{t}, \mathbf{0}] \end{bmatrix} \tag{3.1}$$

where $N = N_{A1} + N_{A2+} + N_{A2-} + N_{B+} + N_{B-}$, and likewise for other vectors. The set $\{b, \boldsymbol{\alpha}_{B+}, \boldsymbol{\alpha}_{B-}\}$ is called the set of free variables, and all other variables are said to be *actively constrained*. We also define the corresponding partitions:

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_{1A} & \mathbf{G}_{A+}^T & \mathbf{G}_{A-}^T & \mathbf{G}_{B+}^T & \mathbf{G}_{B-}^T \\ \mathbf{G}_{A+} & \mathbf{G}_{1B++} & \mathbf{G}_{1B+-}^T & \mathbf{G}_{C++}^T & \mathbf{G}_{C+-}^T \\ \mathbf{G}_{A-} & \mathbf{G}_{1B+-} & \mathbf{G}_{1B--} & \mathbf{G}_{C-+}^T & \mathbf{G}_{C--}^T \\ \mathbf{G}_{B+} & \mathbf{G}_{C++} & \mathbf{G}_{C-+} & \mathbf{G}_{2++} & \mathbf{G}_{2+-}^T \\ \mathbf{G}_{B-} & \mathbf{G}_{C+-} & \mathbf{G}_{C--} & \mathbf{G}_{2+-} & \mathbf{G}_{2--} \end{bmatrix} \tag{3.2}$$

$$\mathbf{H}_2 = \begin{bmatrix} \mathbf{G}_{2++} & \mathbf{G}_{2+-}^T & \mathbf{1} \\ \mathbf{G}_{2+-} & \mathbf{G}_{2--} & \mathbf{1} \\ \mathbf{1}^T & \mathbf{1}^T & 0 \end{bmatrix}$$

where $\mathbf{H}_2$ is assumed to be non-singular for the purposes of this paper (this will always be true if $\mathbf{G}_2$ is non-singular). This partition may change during the completion of the algorithm. We also define the lower triangular matrix $\mathbf{V}$ using:

$$\mathbf{H}_2 = \mathbf{V} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0}^T & -1 \end{bmatrix} \mathbf{V}^T \tag{3.3}$$

While we will not go into detail about how this factorisation of $\mathbf{H}_2$ is used, it should be noted that:

- Using this factorisation, it is possible, given a vector $\mathbf{r}$, to calculate $\mathbf{H}_2^{-1}\mathbf{r}$ in quadratic time.
- If the size of $\mathbf{H}_2$ is increased or decreased when the partitioning is changed, it is possible to update $\mathbf{V}$ in quadratic time.
- The factorisation $\mathbf{V}$ tends to be more numerically stable than some other possible factorisation (for example, $\mathbf{H}_2^{-1}$).

3.2. The Active Set Algorithm

We now outline the algorithm used here to solve the SVM training problem (2.26). Upon entering the algorithm, it is assumed that we have a consistently defined partitioning (3.1) (including $\boldsymbol{\alpha}_{B+}$ and $\boldsymbol{\alpha}_{B-}$ – the starting point), a factorisation $\mathbf{V}$ of $\mathbf{H}_2$ as defined by (3.3), and the gradients $\mathbf{e}$ and f as defined by (2.27).

The algorithm is as follows:

1. Calculate the Newton step on the free variables (assuming that $N_{B+} + N_{B-} \neq 0$):

$$\begin{bmatrix} \Delta\boldsymbol{\alpha}_{B+} \\ \Delta\boldsymbol{\alpha}_{B-} \\ \Delta b \end{bmatrix} = -\mathbf{H}_2^{-1} \begin{bmatrix} \mathbf{e}_{B+} \\ \mathbf{e}_{B-} \\ f \end{bmatrix} \tag{3.4}$$

and calculate:

$$\begin{bmatrix} \Delta\mathbf{e}_{A1} \\ \Delta\mathbf{e}_{A2+} \\ \Delta\mathbf{e}_{A2-} \end{bmatrix} = \begin{bmatrix} \mathbf{G}_{B+}^T & \mathbf{G}_{B-}^T & \mathbf{1} \\ \mathbf{G}_{C++}^T & \mathbf{G}_{C+-}^T & \mathbf{1} \\ \mathbf{G}_{C-+}^T & \mathbf{G}_{C--}^T & \mathbf{1} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{\alpha}_{B+} \\ \Delta\boldsymbol{\alpha}_{B-} \\ \Delta b \end{bmatrix} \tag{3.5}$$

$$\begin{bmatrix} \Delta\boldsymbol{\alpha}_{A1} \\ \Delta\boldsymbol{\alpha}_{A2+} \\ \Delta\boldsymbol{\alpha}_{A2-} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \quad \begin{bmatrix} \Delta\mathbf{e}_{B+} \\ \Delta\mathbf{e}_{B-} \end{bmatrix} = -\begin{bmatrix} \mathbf{e}_{B+} \\ \mathbf{e}_{B-} \end{bmatrix}$$

2. Calculate the scale factor:

$$\beta = \min\left(1, \left.\frac{-\alpha_i}{\Delta\alpha_i}\right|_{\left(\begin{smallmatrix}\Delta\alpha_i\le 0\\ \alpha_i\in\boldsymbol{\alpha}_{B+}\end{smallmatrix}\right) \text{ or } \left(\begin{smallmatrix}\Delta\alpha_i\ge 0\\ \alpha_i\in\boldsymbol{\alpha}_{B-}\end{smallmatrix}\right)}, \left.\frac{-Ct_i - \alpha_i}{\Delta\alpha_i}\right|_{\left(\begin{smallmatrix}\Delta\alpha_i\ge 0\\ \alpha_i\in\boldsymbol{\alpha}_{B+}\end{smallmatrix}\right) \text{ or } \left(\begin{smallmatrix}\Delta\alpha_i\le 0\\ \alpha_i\in\boldsymbol{\alpha}_{B-}\end{smallmatrix}\right)}\right)$$

$$p = \arg\min_i \left(\left.\frac{-\alpha_i}{\Delta\alpha_i}\right|_{\left(\begin{smallmatrix}\Delta\alpha_i\le 0\\ \alpha_i\in\boldsymbol{\alpha}_{B+}\end{smallmatrix}\right) \text{ or } \left(\begin{smallmatrix}\Delta\alpha_i\ge 0\\ \alpha_i\in\boldsymbol{\alpha}_{B-}\end{smallmatrix}\right)}, \left.\frac{-Ct_i - \alpha_i}{\Delta\alpha_i}\right|_{\left(\begin{smallmatrix}\Delta\alpha_i\ge 0\\ \alpha_i\in\boldsymbol{\alpha}_{B+}\end{smallmatrix}\right) \text{ or } \left(\begin{smallmatrix}\Delta\alpha_i\le 0\\ \alpha_i\in\boldsymbol{\alpha}_{B-}\end{smallmatrix}\right)}\right) \tag{3.6}$$

3. Take the step:

$$\{\boldsymbol{\alpha}, b, \mathbf{e}, f\} := \{\boldsymbol{\alpha}, b, \mathbf{e}, f\} + \beta\{\Delta\boldsymbol{\alpha}, \Delta b, \Delta\mathbf{e}, \Delta f\} \quad (3.7)$$

4. If $\beta < 1$:
   a. Re-partition the problem by removing variable p from the set of free variables.
   b. Goto step 1.
5. If the KKT conditions (2.28) are not met:
   a. Calculate:

$$q = \arg\min_i \begin{pmatrix} e_i + \varepsilon_i \left| \begin{pmatrix} \alpha_i \in \boldsymbol{\alpha}_{A1} \\ (\mathbf{x}_i, z_i) \notin \Theta_{R_\le} \end{pmatrix} \text{ or } \begin{pmatrix} \alpha_i \in \boldsymbol{\alpha}_{A2-} \\ (\mathbf{x}_i, z_i) \notin \Theta_{R_\ge} \end{pmatrix} \right., \\ -e_i + \varepsilon_i \left| \begin{pmatrix} \alpha_i \in \boldsymbol{\alpha}_{A2+} \\ (\mathbf{x}_i, z_i) \notin \Theta_{R_\le} \end{pmatrix} \text{ or } \begin{pmatrix} \alpha_i \in \boldsymbol{\alpha}_{A1} \\ (\mathbf{x}_i, z_i) \notin \Theta_{R_\ge} \end{pmatrix} \right. \end{pmatrix} \quad (3.8)$$

   b. Re-partition the problem by adding the variable q to the set of free variables.
   c. Goto step 1.
6. Terminate – the optional solution has been found.

Note that this algorithm will always find the optimal solution in a finite time.

4. INCREMENTAL LEARNING

We have chosen to use a fairly simple heuristic to investigate incremental learning. Suppose we have a trained SVM. If we wish to add more training pairs to our training set, we do so by setting the α's corresponding to these pairs to zero (without changing any other values) and re-entering the algorithm at step 1 with starting point $\{\boldsymbol{\alpha}, b, \mathbf{e}, f\}$.

Likewise, if we wish to remove points from our training set, we simply remove the corresponding α's, update $\mathbf{e}$ and f if necessary (i.e. if we removed a support vector), and then re-enter the algorithm at step 1.

5. EXPERIMENTAL RESULTS

We tested incremental training using a standard Mackey-Glass chaotic time-series [5]. The Mackey-Glass delay differential equation is:

$$\frac{dy(t)}{dt} = -0.1y(t) + \frac{0.2y(t-\Delta)}{q + y(t-\Delta)^{10}} \quad (5.1)$$

where we have chosen $\Delta = 17$. This time-series was simulated using code from the IEEE Working Group on Data Modeling Benchmarks[2]. Based on [6], we constructed our training set with:

$$\mathbf{x}_i = \begin{bmatrix} y(i-1) & y(i-2) & y(i-3) & y(i-4) \end{bmatrix}^T$$
$$z_i = y(i) + e(i) \quad (5.2)$$
$$\text{where: } e(i) \in N(0, 0.01)$$

and the kernel function:

$$K(\mathbf{x}, \mathbf{y}) = \sum_{d=1}^{d_L} \frac{\sin\left(\left(\nu + \frac{1}{2}\right)(x_d - y_d)\right)}{\sin\left(\frac{1}{2}(x_d - y_d)\right)} \quad (5.3)$$

where $\nu = 200$, $C = 10$ and $\varepsilon = 0.001$. Training was done in an incremental fashion, starting with an empty training set and then adding one training pair at a time until 1000 training pairs had been added. After each incremental change, the resultant SVM was tested using an independent testing set consisting of 1000 pairs. The generalization error was measured using the following formula:

$$\sigma^2(f) = \frac{1}{N_{TEST}} \sum_{i=1}^{N_{TEST}} \frac{\left(z(i) - f(\mathbf{x}_i)\right)^2}{\sigma^2(z)} \quad (5.4)$$
$$\text{where: } \sigma^2(z) = \frac{1}{N_{TEST}} \sum_{i=1}^{N_{TEST}} z_i^2$$

Results are shown in figures 1 and 2.

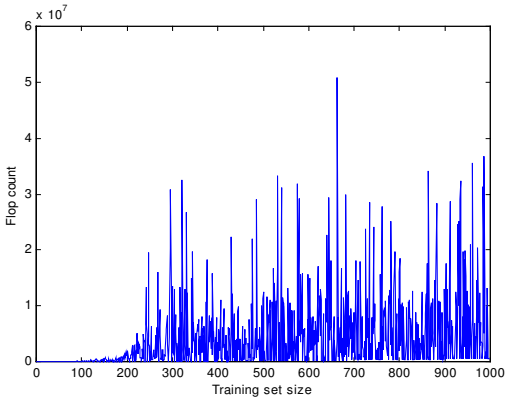

Figure 1 – Incremental flop count per added training point as training set size is increased.

The algorithm was implemented in C++ and compiled with DJGPP. All testing was done in a 1GHz Pentium Coppermine using extended DOS.

[2] http://neural.cs.nthu.edu.tw/jang/benchmark/

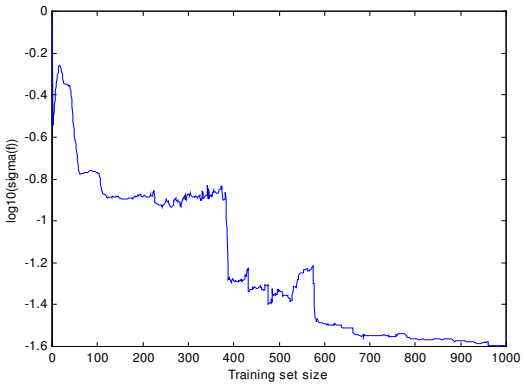

Figure 1 – Machine accuracy as training set size increases.

6. CONCLUSION

We have successfully demonstrated how the problem of pattern recognition and regression may be unified into a single problem. We have also demonstrated the applicability of incremental learning to the regression problem and, by implication, the pattern recognition problem. Results are provided to demonstrate the computational efficiency and advantages claimed.

Acknowledgements: The authors gratefully acknowledge the support from the Australian Research Council and from the Defence Science Technology Organization. Discussions in related works from several colleagues such as Danny Ralph and Brendan Owen are gratefully acknowledged.

7. REFERENCES

[1] C. J. C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition", Knowledge Discovery and Data Mining, 2(2), 1998.

[2] C. Cortes and V. Vapnik, "Support Vector Networks", Machine Learning, Kluwer Academic, Boston, 20:273-297, 1995

[3] R. Fletcher. Practical Methods of Optimisation. John Wiley and Sons, Chichester, 1981.

[4] S. Haykin. Neural Networks – A Comprehensive Foundation. Prentice Hall, New Jersey, 1999.

[5] M. C. Mackey, L. Glass, Science, 197, pp 287-289.

[6] K. -R. Müller, A. J. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, V. Vapnik, "Predicting Time Series with Support Vector Machines", Advances in Kernel Methods – SV learning, 1999, pp 243-254.

[7] V. Vapnik, S. E. Golowich, A. Smola, "Support Vector Method for Function Approximation, Regression Estimation, and Signal Processing", Advances in Neural Information Processing Systems 9, Number 126, Cambridge, MA, 1997, MIT Press, pp 281-287.

[8] A. Fan and M. Palaniswami, "Selecting bankruptcy predictors using a support vector machine approach", International Joint Conference on Neural Networks, Como, Italy, July 24-27, 2000.

[9] A. Shilton, M. Palaniswami, D. Ralph, and A.C. Tsoi, Incremental Training in Support Vector Machines, in Proc. of the International Joint conference on Neural Networks, Washington, 2001.

[10] M. Palaniswami, A. Shilton, D. Ralph, and B. Owen – "Machine leanring using support vector machines", International conference on Artificial Intelligence in Science and Technology, Hobart, Australia, December 2000.

[11] B. Owen, M. Palaniswami and J. Harris – "Automated Monitoring of Fishways", in Proc. of 3rd IEEE International Conference on Intelligent Processing Systems, ICIPS, pp. 238-242, Gold Coast, Australia, August 1998.