This is a repository copy of *An Online Support Vector Learning Method*.

White Rose Research Online URL for this paper:
http://eprints.whiterose.ac.uk/84343/

Monograph:
Drezet, Pierre.M.L. and Harrison, R.F. (2000) An Online Support Vector Learning Method. UNSPECIFIED. ACSE Research Report 771 . Department of Automatic Control and Systems Engineering

---

Reuse
Unless indicated otherwise, fulltext items are protected by copyright with all rights reserved. The copyright exception in section 29 of the Copyright, Designs and Patents Act 1988 allows the making of a single copy solely for the purpose of non-commercial research or private study within the limits of fair dealing. The publisher or other rights-holder may allow further reproduction and re-use of this version - refer to the White Rose Research Online record for this item. Where records identify the publisher as the copyright holder, users can verify any specific terms of use on the publisher's website.

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

An online support vector learning method

Pierre M. L. Drezet and Robert F. Harrison

Department of Automatic Control and Systems Engineering
University of Sheffield
Mappin Street, Sheffield, UK, S1 3JD

Research Report: 771

June 21, 2000

Abstract

An online iterative implementation of support vector(SV) learning is presented. The method converges to the SV solution for batch data and results in a minimal support vector set. For the online case, convergence to the SV solution holds for stationary data, and additional automatic methods to control regularisation are demonstrated. For online applications, methods to bound the number of support vectors in an optimal manner are described. Demonstrations of the method for large data set discrimination and the prediction of chaotic dynamic systems and are included.

UNIV OF SHEFFIELD LIBRARY ST. GEORGE'S

200597150

1 Introduction

Support Vector Machines (SVMs) are a group of related methods in a common frame-work for inductive learning. Classification and regression can be carried out within this frame-work, based on utilising training vectors to span the *feature space* of the approximation function. The description of a function's input space as a 'feature space' comes about because non-linear mappings of input patterns into a higher dimensional space are used to generalise the linearity assumptions that form the basis of the support vector (SV) frame-work.

SV methods use support vectors to build a *data-dependent* representation of a vector of weights as for those in single layer feed-forward linear networks. This representation involves only dot products of training vectors and permits the replacement of these terms by kernel functions, allowing non-linear functions of input patterns to be represented by higher dimensional linear functions.

The cost function usually optimised in SV learning includes a regularisation term in addition to the application dependent training error cost, $\xi(e)$, where $e_i = y_i - f(x_i)$.

$$J(\vec{w}) = ||\vec{w}||^2 + C\xi(e) \tag{1}$$

$||\vec{w}||$ is the l_2 norm of the weight vector associated with each input feature. For classification the error cost is chosen to be insensitive to pattern vectors that are classified with a comfortable margin, which when combined with regularisation gives the *optimal margin*. Labeling target values, $y \in -1, 1$ gives an error function.

$$\xi(e_i) = \begin{cases} y_i e_i & \text{for } y_i e_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

A similar discontinuous error cost is conventionally used for regression, this being Vapnik's ϵ-insensitive error loss.

$$\xi(e_i) = \begin{cases} |e_i| - \epsilon & \text{for } |e_i| > \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where $|e|$ is the l_1 magnitude of the prediction error.

These discontinuous loss functions require constrained optimisation methods to be used, resulting in Karush-Khun-Tucker (KKT) conditions in the optimisation. These conditions provide a means for the dual formulation to yield a sparse solution. Without such constraint conditions the (often) semi-definite cost objective leads to a particular solution expanded on all training vectors.

These types of error loss functions are not, however, optimal for many types of noise distributions encountered in empirical data, but do have some robust properties [13, 10]. The KKT conditions result in each training vector that causes non-zero error functions to be included in the support vector set. A draw-back of this criterion is that the support vector set becomes over-specified, and the temptation for the practitioner is to choose values of C and ϵ to control the number of support vectors to a manageable size at the expense of predictive performance. For the case of regression it is normally desirable to set ϵ to zero so that undue bias in the model is avoided, but this results in all training vectors becoming support vectors.

Sparsity controlled formulations [3, 5, 4] of the generic SVM objectives can be used to optimise the number of support vectors that result, but these are still based on constrained quadratic optimisation of batch training data, and are not therefore, practical for online applications.

The training method presented here is based on the simple iterative method of gradient descent - a generalisation of the ADALINE for non-linear problems. The *Kernel–ADALINE* [6] minimimises the continuous cost function of least squares error loss, computed in a variable space of support vector multipliers, but simultaneously optimises equation 1 in the larger variable space of $\vec{w}$. This direct algorithm is, however, unsuitable for online purposes because there is no sparsity in the resulting support vector set, the entire training set is required by the approximation function. A related method, the *Kernel–ADATRON* [8], for classification uses gradient descent with an additional update rule that constrains variable updates and converges to give a similar support vector set to the SV method. Again this is not readily suitable for on-line applications as it is intended for use with multiple training data presentations to compact the support vector set.

Our adaptation of gradient descent for online applications uses the property of the linear *feature space* that is induced by the kernel functions, where a linearly independent support vector set is constructed. Each vector is sequentially projected on to the existing set of training vectors and its update value is apportioned among the existing SV multipliers. The technique for calculating projections in feature space is that of the set reduction algorithm [7], where the linearity of the feature space is calculated by exploitation of the *kernel correlation matrix.*

Details of the kernel–ADALINE are reviewed in section 2, and the set reduction algorithm in section 3. Some benchmark examples and adaptive filtering applications are given in section 7.

2 Kernel-ADALINE

The kernel–ADALINE [6] attempts to solve regression problems in a similar fashion to the ADALINE, i.e. to find iteratively a linear model with the least squares error. Simply minimising the training error, where a nonlinear model is used, leads to *over-fitting* and hence a regularisation term is recommended to control complexity in the model. A popular method of regularisation in iterative linear learning methods is weight decay, which when formulated as in equation 4 leads to Ridge Regression, [9] where the l_2 norm of the weight vector $\vec{w}$ is included as a cost.

$$\vec{w}_{new} \leftarrow (1 - \eta\rho)\vec{w}_{old} + \eta\vec{x}_i e_i \tag{4}$$

and e_i is the prediction error

$$e_i = y_i - < \vec{w}_{old}, \vec{x}_i > \ ^1 \tag{5}$$

For this linear modeling technique the cost function is similar to the SV cost function, equation 1, where a squared error cost is chosen for ξ. By inspection of equation 4 it is noted that the weight vector can be written as a linear combination of training vectors.

$$\vec{w} = \sum_{j=1}^{l} \alpha_j \vec{x}_j \tag{6}$$

Generalising the iterative process (4) for nonlinear approximation functions is possible by substitution of the weight terms with this *data-dependent* weight representation, (6), and substituting Mercer kernels [13] to represent the dot product. The kernel function now represents a dot-product in a Hilbert space, H, mapped into by a non-linear operator $\vec{\phi}$.

$$\sum_{j=1}^{l} \alpha_j^{new} \vec{\phi}(\vec{x}_j) \leftarrow (1 - \eta\rho) \sum_{j=1}^{l} \alpha_j^{old} \vec{\phi}(\vec{x}_j) + \eta\vec{\phi}(\vec{x}_i) e_i \tag{7}$$

which by manipulation leads to [2]

$$\alpha_i^{new} \leftarrow (1 - \eta\rho)\alpha_i^{old} + \eta e_i \tag{8}$$

[1] A bias term is not included for simplicity. A constant term can be augmented to $\vec{\phi}(x_i)$, or a separate bias update used where regularisation of this variable is not desired.

[2] This is just one possible formulation if $\vec{\phi}(\vec{x}_i)$ is not linearly independent for all i, leading to min. norm solution

This update does not include any direct reference to the nonlinear mapping, $\vec{\phi}$, apart from the calculation of the error, e, where a dot product form suitable for substitution with a kernel function, K, is found,

$$\begin{aligned} f(\vec{x}_i) &= <\sum_{j=1}^{l} \alpha_j \vec{\phi}(\vec{x}_j), \vec{\phi}(\vec{x}_i)> & (9) \\ &= \sum_{j=1}^{l} \alpha_j <\vec{\phi}(\vec{x}_j), \vec{\phi}(\vec{x}_i)> & (10) \\ &= \sum_{j=1}^{l} \alpha_j K(\vec{x}_j, \vec{x}_i) & (11) \end{aligned}$$

The kernel ADALINE algorithm thus requires a prediction to be made for each training vector, $\vec{x}_{1..l}$, and a simple update of α_i. The training data is then re-presented as many times as necessary for convergence. Efficiency considerations suggest a *cache* matrix of kernel function values calculated for all i, j, to avoid repetitive kernel calculations. The resulting approximation function is thus a linear combination of *all* training examples in feature space, and as such, at each iteration, all training points must be known.

3 Set-Reduction

A method to reduce the resulting number of support vectors which has been applied to SVMs using the conventional dual formulation, and as we have seen, would be particularly applicable to the kernel ADALINE, is the *set-reduction* algorithm [7, 11]. This post-training process attempts to remove redundancy in the support vector set by looking to the feature space for linear dependency of support vectors. Direct reference to the feature space, H is avoided as usual by the application of Mercer kernels. The concept is as follows using the notation $\vec{z} = \vec{\phi}(\vec{x})$ to represent vectors in feature space, H, of dimension m.

1. Choose an un-tested support vector $\vec{x}_{test}$ and attempt to solve the simultaneous equation to find multipliers m_i for the remaining support vectors $x_{1,\ldots,l-1}$ [3]

$$\begin{array}{ccccccccc} m_1 z_{1,1} & + & m_2 z_{2,1} & + & \ldots & + & m_{l-1} x_{l-1,1} & = & \vec{x}_{test,1} \\ m_1 z_{1,2} & + & m_2 z_{2,2} & + & \ldots & + & m_{l-1} z_{l-1,2} & = & \vec{x}_{test,2} \\ \vdots & + & \vdots & + & \vdots & + & \vdots & = & \vdots \\ m_1 z_{1,p} & + & m_2 z_{2,p} & + & \ldots & + & m_{l-1} z_{l-1,p} & = & \vec{x}_{test,p} \end{array} \quad (12)$$

2. If a solution exists then update each support vector multiplier α_i with $m_i \alpha_{test}$ repeat from 1 and remove the support vector $\vec{x}_{test}$.

[3] The system of equations will not necessarilly be unique for the post processing set reduction algorithm, however, any particular solution will do.

To implement this concept, the linearity of the dot-product Mercer kernel substitution allows us to write the following feature space correlation matrix.

$$\begin{array}{ccccccccc} m_1 < \vec{z}_1, \vec{z}_1 > & + & m_2 < \vec{z}_1, \vec{z}_1 > & \ldots & m_{l-1} < \vec{z}_1, \vec{z}_{l-1} > & = & < z_1, z_{test} > \\ m_1 < \vec{z}_1, \vec{z}_2 > & + & m_2 < \vec{z}_2, \vec{z}_2 > & \ldots & m_{l-1} < \vec{z}_2, \vec{z}_{l-1} > & = & < z_2, z_{test} > \\ \vdots & + & \vdots & \vdots & \vdots & = & \vdots \\ m_1 < \vec{z}_1, \vec{z}_p > & + & m_2 < \vec{z}_2, \vec{z}_p > & \ldots & m_{l-1} < \vec{z}_{l-1}, \vec{z}_p > & = & < z_p, z_{test} > \end{array} \tag{13}$$

The multipliers, m_i, can be found from this re-formulated linear set of equations using pivoting techniques. The feature space dimension, m, is often greater than the number of training vectors, l, and a solution of the systems 12 or 13 may not exist, hence the full set of training vectors must be retained. In practice however many of the support vectors will be approximately co-linear and the pivoting process can be assumed to give a satisfactory result even if a *small* residual pivot value remains.

4 Online SVM Algorithm

The online algorithm uses the gradient descent update rule to calculate α_i, and uses an efficient on-line set reduction algorithm to maintain a compact set of support vectors of number l. An outline of the algorithm for each new example vector $\vec{x}_{new}$ is:

1. Calculate multiplier update, α_{new} using equation 8,
2. Perform weight decay in all other α_i,
3. Augment support vector set with new example vector: $\mathbf{x} = [\mathbf{x} \quad \vec{x}_{new}]$,
4. Produce kernel correlation matrix $\mathbf{K} = \{K(\vec{x}_i, \vec{x}_j) \text{ for } i, j = 1, \ldots, l\}$,
5. Try to solve for $\vec{m}$ in equations 13, where the new vector $\vec{x}_{new}$ is the test vector $\vec{x}_{test}$,
6. If no solution, add training vector $\vec{x}_{new}$ to SV set and multiplier α_{new} to $\vec{\alpha}$, otherwise update all existing α_i by adding $m_i \alpha_{new}$.

The steps 4 and 5 involve many repetitive calculations that can be removed from the algorithm. In step 4 the kernel matrix need only be re-calculated for $K(\vec{x}_i, \vec{x}_{new})$, $i = 1, \ldots, l$, i.e. l kernel function operations per training vector. Step 5 can be optimised by storing the kernel matrix in triangular form, such that only the additional row of l matrix elements is

eliminated during pivoting. A row manipulation history of pivot values can be stored in the zero part of the triangular matrix so that the additional column of kernel functions involving $\vec{x}_{new}$ is also transformed into the triangular representation. The solution to the problem of calculating $\vec{m}$ from the triangular matrix is finally obtained by back substitution. The incremental triangulation procedure requires the number of pivoting computations to grow quadratically with the number of support vectors, rather than as a cubic function. Details of the optimised algorithm are found in the appendix.

For dichotomising, the α update can be simply adjusted by substituting e_i in equation 8 for $\xi(e_i)$ from equation 2. This partially insensitive error update results in many of the error costs equaling zero, and hence the steps 3 to 6 can be ignored leading to increased computational efficiency. Insensitivity to training vectors that are classified with a *large margin* allows convergence to the SVM classifier cost function where only borderline training vectors are considered. Near convergence, the training vectors that create a non-zero update value are the same as those that are included in the support vector set of conventional SVMs. The support vector set of the online algorithm is of course different because additional criteria is used to choose which training vectors each update is shared amongst.

5 Parameter tuning

The parameters of the online method are learning rate, weight decay rate, and kernel parameter. Methods of controlling the learning rate, η, can be borrowed directly from other on-line techniques, but care must be taken that the the duality of weight decay in $\vec{\alpha}$ and $\vec{w}$ space is maintained. The weight decay parameter, ρ, can also be regulated on-line by use of new vectors as validation data, on the assumption that the training data is not re-presented. A method to adjust the weight decay parameter relies on measurement of prediction bias, estimated by the correlation of error and target y_i.

$$\psi_i = y_i e_i \tag{14}$$

A bound for an unbiased, unregularised, prediction function is $\bar{\psi} = 0$, where the notation $\bar{x}$ represents the mean value. The other extreme for over regularisation, $||w|| \to 0$ is $\psi \to y^2$. From this rationale the level of regularisation can be controlled to maintain the bias close to zero,

$$\rho \leftarrow \rho - \frac{\bar{\psi}}{\bar{y^2}}\delta \tag{15}$$

subject to $0 < \rho < 1$ to maintain valid ranges of ρ. $\bar{y^2}$ and $\bar{\psi}$ are average values that may be taken over a *recent* window of time or a more computationally efficient moving average approximation. The $\bar{y^2}$ term normalises the update between the extreme condition where $||w|| = 0$ and ρ should be updated by the maximum increment δ to towards zero. This update rule assumes absolute precision is a requirement as a consequence of $\bar{\psi} = 0$, and regularisation will only increase as a result of random perturbation of $\bar{\psi} < 0$. Knowledge of the amount of noise present in the sample data and the required prediction precision, often however, allows for increased regularisation to decrease error variance.

In non-stationary environments, weight decay regularisation has a more complex role. Weight decay effects the *long term memory* of the model, introducing bias toward learning more recent system behaviour. Adaptive filtering methods often refer to the rate of weight decay as the *forgetting factor*. This on-line regularisation has an opposing effect to that in the sense of batch data where local specialisation is a preferably avoided. batch SV methods use regularisation specifically to avoid specialisation, but here in the online case the same type of regularisation has the opposite effect. Local specialisation is often desirable for non-stationary systems and exaggerated regularisation may be appealing to enhance the rate of adaptation to real-time variations.

To generalise the equation 15 for over-regularisation a parameter β can be introduced,

$$\rho \Leftarrow \rho - \left(\frac{\bar{\psi}}{\bar{y^2}} - \beta \right) \delta \tag{16}$$

subject to $0 < \rho < 1$, and where $0 < \beta < 1$ would be nominal values.

It is convenient to control the dynamic of $\bar{\psi}$ and $\bar{y^2}$ by a moving average estimate. The time constant of this average and the update gain δ, defines the dynamic of the regularisation process. Short time constants allow the system to *forget* learned behaviour quickly. The applicability of very long time constants is not obvious, because regularisation will only come into effect after considerable data has been learned and the requirement for regularisation is lost. The intermediate stage can be considered as requiring regularisation for the same reasons as for batch data, to minimise the weight norm where insufficient training data has been presented. The update rule automatically decreases weight decay at very early stages of training where regularisation is automatic, assuming values of α are initially zero.

6 Online SV set control

Online applications of this method have the drawback that where kernel functions such as Gaussians are used, the feature space dimension is infinite. Consequently there is theoretically no bound on the number of support vectors that may be required [4], and this may lead to computational problems of storage even if an artificially low numerical precision, is used to select the most diverse SVs. The options to limit the size of the SV set are:

1. Stop including new support vectors at some limit, and update existing parameters, α, by sharing approximately.

2. Where some upper limit is reached, exchange new SVs with ones with the lowest value of α, (excluding recently introduced SVs).[12]

3. Maintain a record of the smallest angle (or max cosine) each new SV, $\vec{x}_{new}$, makes in feature space with any other, $\vec{z}_i$, i.e. $\max\left(\frac{<\vec{z}_{new},\vec{z}_i>}{||\vec{z}_{new}||||\vec{z}_i||}\right)$. When some upper limit is reached, new support vectors are exchanged for the most dependent existing vector.

Option 1 is unappealing in most online applications because stationarity is assumed. The second option depends on the *usage* of the support vector. An assumption of this method is that the lowest values of α_i have least contribution to the approximation function, which is not always appropriate because the associated support vector may yield large dot products. The last option, 3, attempts to maintain an SV set that has the most independent spanning set in feature space. These angles can be calculated using kernel functions and stored for each support vector. This last option is most appealing from a theoretical point of view because a near *optimal* set of support vectors is maintained. In practice, however, the addition of a usage factor such as option 2 helps remove support vectors that are independent but 'outlying' because of input space noise. For options 2 & 3 there is an additional computational burden for exchanging support vectors because of triangulation during pivoting. As a result the cache matrix must be re-triangulated from whichever row the support vector was removed from, each time an SV is exchanged.

This substitution method 2 could also be used to find a near orthogonal basis in feature space. The procedure would requires the data distribution to be sufficiently dense in feature space that a near orthogonal set exists amongst the examples. A full basis for the feature sapce would obviously require many more training examples then the dimension of feature space.

[4]Numerical precision and data distribution are factors that determine the number of support vectos in practice

7 Examples

7.1 Driven pendulum: On-line

The chaotic behaviour of a pendulum being forced by a sufficiently large torque signal, u, has been simulated to show the improved tracking capabilities of a nonlinear adaptive filter compared to the linear case. The system to be predicted is given by the differential equation

$$\ddot{\Theta} + 0.05\dot{\Theta} + 9.8 \sin \Theta = u \tag{17}$$

The measured value to be predicted is the angular velocity. Figure 7.1 shows an interval where the dynamic behaviour changes as the periodic behaviour changes during full rotations. In comparison to the nonlinear adaptive filter, a linear adaptive filter is demonstrated on the same data interval. The linear filter uses an equivalent LMS update rule to the nonlinear filter, but only for linear terms i.e.

$$\vec{w}_{new} \leftarrow \vec{w}_{old} + \eta \vec{x} e \tag{18}$$

where e is the prediction error defined as in equation 5. Both filters use 100 time lagged samples with 1 second interval. The learning rate η is the highest value that maintained stability. The kernel chosen for the online algorithm is a 3_{rd} order polynomial $(1+ < \vec{x}, \vec{y} >)^3$.

The prediction of the linear filter is seen to take longer to re-converge at points where dynamic behaviour changes rapidly, even with the maximum step length update. In comparison, the nonlinear filter adapts rapidly to the new dynamic properties. This property may be a result of a non-linear adaptive model converging close to a global non-linear model rather than the local linear models of LMS, and adaptation to new behaviour resulting from numerical sensitivity is therefore rapid.

7.2 Large data sets: Off-line

To demonstrate the computational abilities of the online SVM when learning a large batch of classification data, two examples have been chosen: One example is a large scale, but easy, synthetic problem and the other is a hard problem from real world data. The on-line algorithm was implemented in the C language and compiled with gcc [1].

7.2.1 Low dimensional synthetic problem

A two dimensional, two class, and overlapping Gaussian distributed pattern space consisting of 120,000 samples was generated. The distribution and

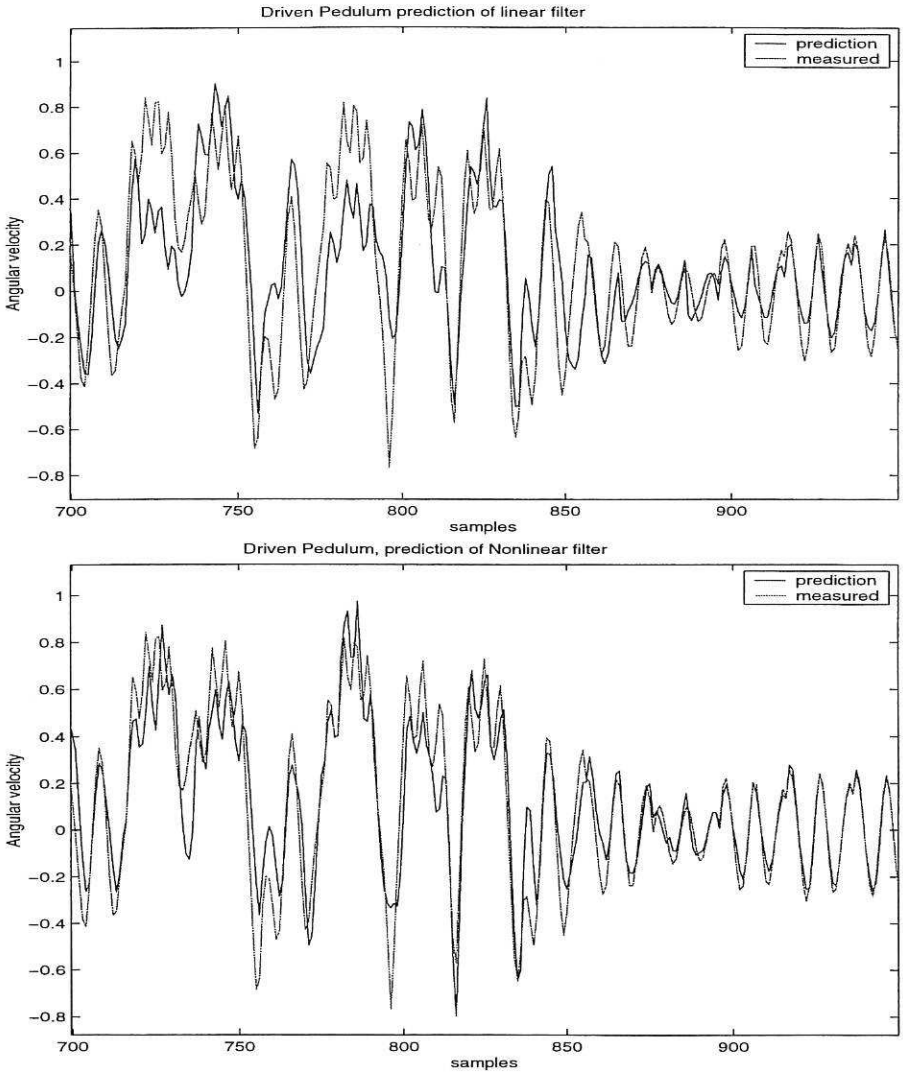

Figure 1: 10 step ahead prediction of pendulum angular velocity using linear (top) and nonlinear (bottom) adaptive filter

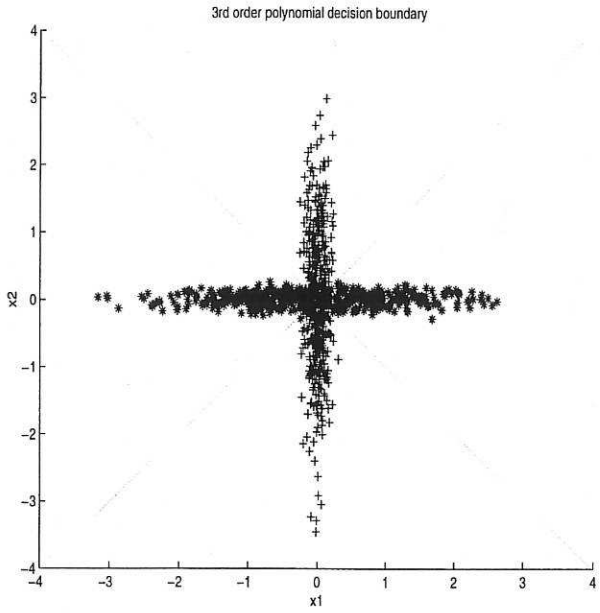

Figure 2: Synthetic 2-class pattern space. 120,000 training examples (equal numbers of each class).

resulting decision boundary of the online SVM is shown in figure 2.

The on-line algorithm tool completed in less than 30 seconds on a SUN micro-computers 160MHz Sparc Ultra-1 and required less than 10 Kbytes of RAM. The resulting decision boundary overlaps the theoretical optimal boundary shown in figure 2 This problem, though simple, would require a quadratic programming problem of 120,000 variables when solved by conventional SVM formulations. The kernel used was 3^{rd} order polynomial of the form, $(1+ < x, y >)^3$, and so for the two dimensional problem the feature space dimension was limited to 10, hence only 10 support vectors are required. This small number of SVs is responsible for the small computational requirements of the algorithm.

7.2.2 High dimensional real problem

Optical character recognition benchmark data from the MNIST database has been chosen for a large scale, large dimensional problem [2]. The data set consists of 60,000 images of the numbers $0, \ldots, 9$ in 28 by 28 pixel format. The images in the database are already centered and normalised and are used directly as the pattern space of the classifier. In this report we consider only the binary classification problem of 1 character from the remaining 9 digits,

# SVs	Computation time	Overall(%)	Sensitivity(%)	Specificity(%)
139	40 mins.	97.8	82.8	91.2

Table 1: Results for MNIST Optical hand written character recognition data set

namely the number 5. The emphasis of this preliminary benchmark test is on the computational effort rather than the exact final test set performance. The number of support vectors was restricted to 1024, and a 3^{rd} order polynomial kernel was used which gives a feature space dimension of 1.8×10^5. Memory requirements were 16MBytes (8 byte floating point format) on the above machine.

The resulting predictive error for the test set is comparable with other results [2]. The computation time can be seen to be sensitive to the number of support vectors allowed. This data set will be fully explored with a multi-class online algorithm in forthcoming work.

8 Conclusions

The online SV method is a simple self organising adaptive nonlinear filter. It has many of the advantages of linear adaptive filters such as convexity and stability, and also properties of highly nonlinear processes such as self organisation and optimisation of nonlinear functions. The adaptive system is also capable of batch training, providing conventional SV results, but often resulting in a sparser SV set. The algorithm is simply adapted for dichotomising functions and results in increased learning efficiency over the regression algorithm. The iterative nature of the learning procedure is similar to many other neural network approaches where useful techniques such as early stopping using validation data can be applied.

The online algorithm used for batch data learning is highly efficient in problems where the number of SVs is much smaller than the number of training examples because of the large decrease in the number of optimisation variables. The computation time is quadraticaly sensitive to the absolute number of support vectors, but the smaller memory requirements (compared to other SVM and kernel ADALINE, ADATRON) make the online method highly competitive in terms of computational effort.

A Detailed optimised online algorithm

The algorithm for each new example vector

1. if $|x| = 0$ ignore and go to next
2. Calculate error: $e = f(x) - y$
3. Calculate update: $u = ne$
4. Augment $\mathbf{H}$ with $K(\vec{z}_i, \vec{x})$
5. LOOP (j) through all but the last row of $\mathbf{H}$
6. LOOP (i) from beginning of row j to j^{th} column of H
7. Update new element of column j, $H_{l,j} = H_{l,j} - H_{j,i}H_{j-i,l}$
8. END LOOP
9. END LOOP (new column of H is fixed)
10. LOOP (j) through all but the last row of $\mathbf{H}$
11. Do weight decay: SV(j)= $(1 - \eta\rho)SV(j)$
12. LOOP (k) from j_th to end of j^{th} column of H
13. Find pivot: p=$\mathbf{H}_{k,j}/\mathbf{H}_{j,j}$
14. Triangulate vectors: $\mathbf{H}_{k,1..l} = \mathbf{H}_{k,1..l} - p\mathbf{H}_{j,1..l}$
15. Store pivot value $H_{j,k-j} = p$.
16. END LOOP
17. END LOOP (H is now re-triangulated)
18. IF last row of triangulated matrix=0 (new vector is dependent)
19. LOOP (i) through triangular matrix
20. LOOP (j) from last row of triangulated matrix to i^{th} row
21. Subtract knowns from last colum:n $\mathbf{H}_{l-i,l} = \mathbf{H}_{l-i,l} - m_j\mathbf{H}_{l-i,l-j}$
22. END LOOP

23. Calculate adjustment: $m_i = \mathbf{H}_{l-i,l}/\mathbf{H}_{l-i,l-i}$

24. Update multipliers: $\alpha_{l-i} = \alpha_{l-i} + m_i u$

25. END LOOP (new update has been shared out)

26. Remove example vector from support vector set.

27. ELSE

28. Leave new support vector and augment $\vec{\alpha}$ with u.

29. END IF (new support vector added)

B $l2$ weight norm and the prediction bias

For a fractional decrease in $||\vec{w}||_2^2$ using weight decay in an iterative update rule $\vec{w} \Leftarrow (1-\rho)\vec{w}$, the approximation function $f(\vec{x}) =< \vec{w}, \vec{x} >$ is reduced in magnitude, as a result of dot product linearity.

References

[1] http://www.gnu.org/software/software.html.

[2] http://www.kernel-machines.org/.

[3] P. Drezet and R. F. Harrison. Directly optimised support vector machines for classification and regression. Research Report 716, University of Sheffield, Dpt. Automatic Control & Systems Engineering, 1998.

[4] P. Drezet and R. F. Harrison. An efficient formulation of sparsity control support vector machines. In *Proceedings, 7th European Symposium of Artificial Neural Networks*, pages 207–212. Elsevier, April 1999.

[5] P. Drezet and R. F. Harrison. Sparsity controlled support vector machines. *Pattern Recognition*, In Press.

[6] T-T. Frieß and R.F. Harrison. The kernel adaline: A new algorithm for non-linear signal processing and regression. Research Report 731, University of Sheffield, Dpt. Automatic Control & Systems Engineering, 1998.

[7] T-T. Frieß and R.F. Harrison. Linear programming support vector machines for pattern classification and regression estimation; and the sr algorithm: Improving speed and tightness of VC bounds in SV algorithms. Research Report 706, University of Sheffield, Dpt. Automatic Control & Systems Engineering, 1998.

[8] T-T. Frieß and R.F. Harrison. Support vector neural networks. Research Report 725, University of Sheffield, Dpt. Automatic Control & Systems Engineering, 1998.

[9] A. E. Hoerl and R. W. Kennard. Ridge regression: biased estimation for nonorthogonal problems. *Technometrics*, 12:55–67, 1970.

[10] P. Huber. Robust estimation of location parameter. *Annals of Mathematical Statistics*, 1(20), 1964.

[11] B. Schölkopf, S. Mika, C.J.C. Burges, P. Knirsch, K.-R. Müller, G. Rätsch, and A. Smola. Input space vs. feature space in kernel-based methods. *IEEE Transactions on Neural Networks*, 10(5):1000–1017, 1999.

[12] J. A. K. Suykens, L. Lukas, and J. Vandewalle. Sparse least squares support vector machine classifiers. In *ESANN'2000 European Symposium on Artificial Neural Networks*, pages 37–42, 2000.

[13] V. N. Vapnik. *The nature of statistical learning theory.* Springer-Verlag New York, 1995.

UNIV OF SHEFFIELD LIBRARY ST. GEORGE'S