ISIS Technical Report

Support Vector Machines for Classification and Regression

Steve Gunn

10 November 1997

Image Speech & Intelligent Systems Group
University of Southampton

Contents

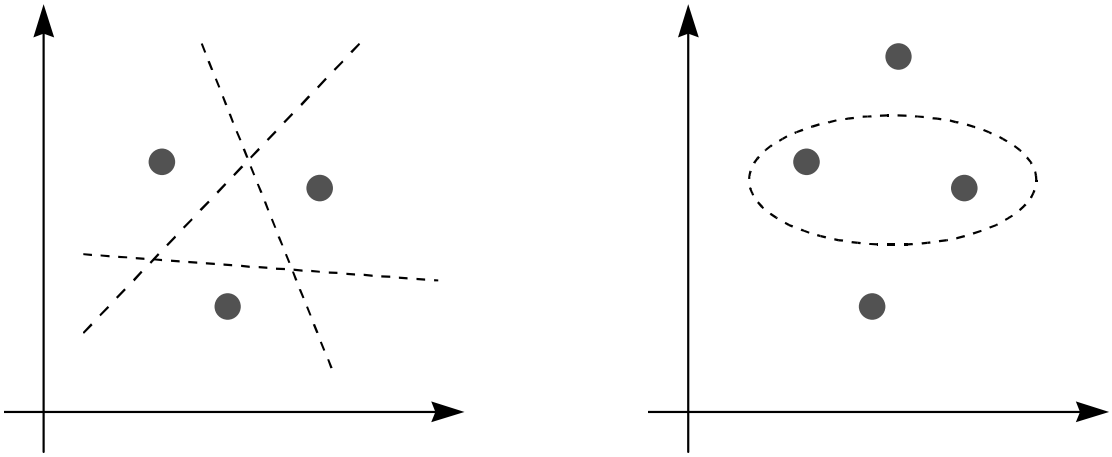
1 Introduction

The foundations of Support Vector Machines (*SVM*) have been developed by Vapnik [19] and are gaining popularity due to many attractive features, and promising empirical performance. The formulation embodies the Structural Risk Minimisation (*SRM*) principle, which in our work [8] has been shown to be superior to traditional Empirical Risk Minimisation (*ERM*) principle employed by conventional neural networks. *SRM* minimises an upper bound on the generalisation error, as opposed to *ERM* which minimises the error on the training data. It is this difference which equips *SVM*s with a greater ability to generalise, which is our goal in statistical learning. *SVM*s were developed to solve the classification problem, but recently they have been extended to the domain of regression problems [18].

In the literature the terminology for *SVM*s is slightly confusing. The term *SVM* is typically used to describe classification with support vector methods and support vector regression is used describe regression with support vector methods. In this report the term *SVM* will refer to both classification and regression methods, and the terms Support Vector Classification (*SVC*) and Support Vector Regression (*SVR*) will be used for specification.

The report starts with an introduction to the structural risk minimisation principle. The *SVM* is introduced in the setting of classification, being both historical and more accessible. This leads onto mapping the input into a higher dimensional feature space by a suitable choice of kernel function. The report then considers the problem of regression. To illustrate the properties of the techniques two examples are given.

The VC dimension is a scalar value that measures the capacity of a set of functions.



The set of linear indicator functions has a VC dimension equal to n+1. The figure illustrates how three points in the plane can be shattered by the set of linear indicator functions whereas four points cannot. In this case the VC dimension is equal to the number of free parameters, but in general that is not the case. e.g. the function Asin(bx) has an infinite VC dimension.

SRM principle creates a structure

2 Support Vector Classification

The classification problem can be restricted to consideration of the two class problem without loss of generality. In this problem the goal is to separate the two classes by a function which is induced from available examples. The goal is to produce a classifier which will work well on *unseen* examples, i.e. it generalises well. Consider the example in Figure 1. Here there are many possible linear classifiers that can separate the data well, but there is only one which maximises the margin (maximises the distance between it and the nearest data point of each class). This linear classifier is termed the optimal separating hyperplane. Intuitively, we would expect this boundary to generalise well as opposed to the other possible boundaries.

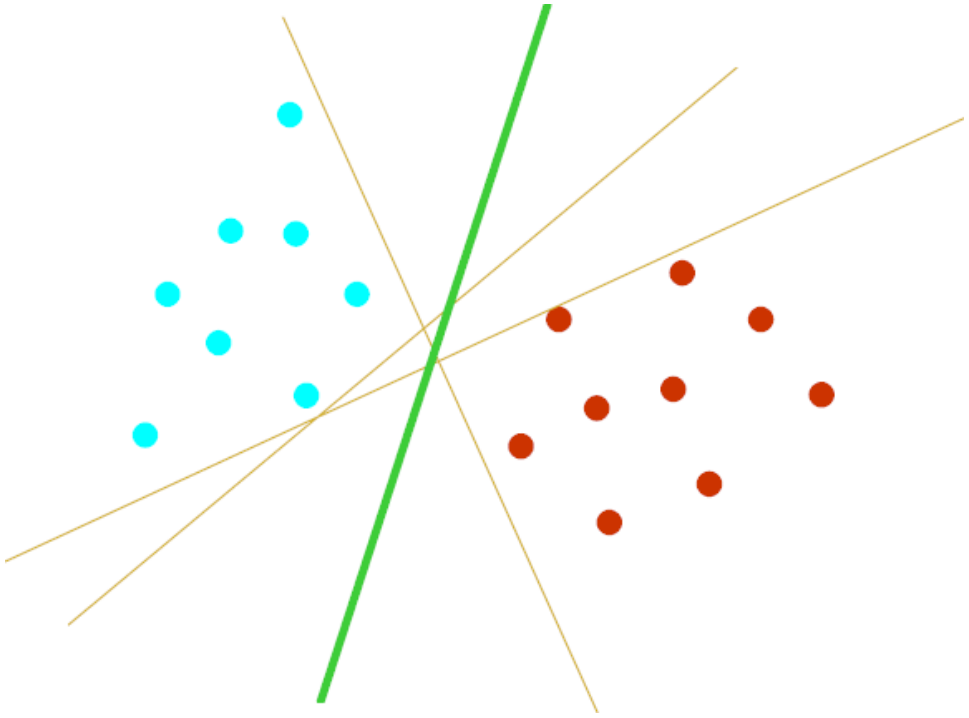


Figure 1 *Optimal Separating Hyperplane*

2.1 The Optimal Separating Hyperplane

Consider the problem of separating the set of training vectors belonging to two separate classes.

$$(y_1, \mathbf{x}_1), \ldots, (y_l, \mathbf{x}_l), x \in R^n, y \in \{-1, +1\} \tag{1}$$

with a hyperplane

$$(\mathbf{w} \cdot \mathbf{x}) + b = 0 \tag{2}$$

The set of vectors is said to be *optimally separated* by the hyperplane if it is separated without error and the distance between the closest vector to the hyperplane is maximal. There is some redundancy in Equation (2), and without loss of generality it is appropriate to consider a canonical hyperplane [19], where the parameters **w**, *b* are constrained by,

$$\min_{\mathbf{x}_i} |\mathbf{x} \cdot \mathbf{w} + b| = 1 \tag{3}$$

This incisive constraint on the parameterisation is preferable to alternatives in simplifying the formulation of the problem. In words it states that*: the norm of the weight vector should be equal to the inverse of the distance, of the nearest point in the data set to the hyperplane*. The idea is illustrated in Figure 2.

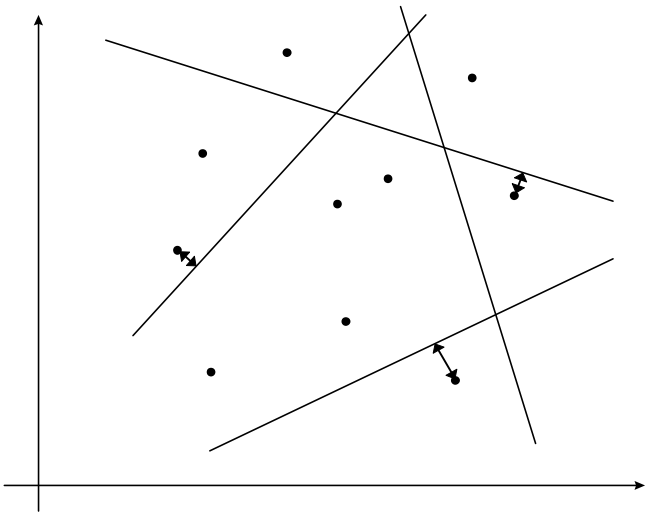


Figure 2 *Canonical Hyperplanes*

A separating hyperplane in canonical form must satisfy the following constraints,

$$y_i \left[(\mathbf{w} \cdot \mathbf{x}_i) + b \right] \geq 1, \quad i = 1, \ldots, l \tag{4}$$

The distance $d(\mathbf{w}, b; \mathbf{x})$ of a point $\mathbf{x}$ from the hyperplane $(\mathbf{w}, b)$ is,

$$d(\mathbf{w}, b; \mathbf{x}) = \frac{|\mathbf{w} \cdot \mathbf{x} + b|}{\|\mathbf{w}\|} \tag{5}$$

The optimal hyperplane is given by maximising the margin, $\rho(\mathbf{w}, b)$, subject to the constraints of Equation (4). The margin is given by,

$$\begin{aligned} \rho(\mathbf{w}, b) &= \min_{\{\mathbf{x}_i : y_i = 1\}} d(\mathbf{w}, b; \mathbf{x}_i) + \min_{\{\mathbf{x}_j : y_j = -1\}} d(\mathbf{w}, b; \mathbf{x}_j) \\ &= \min_{\{\mathbf{x}_i : y_i = 1\}} \frac{|\mathbf{w} \cdot \mathbf{x}_i + b|}{\|\mathbf{w}\|} + \min_{\{\mathbf{x}_j : y_j = -1\}} \frac{|\mathbf{w} \cdot \mathbf{x}_j + b|}{\|\mathbf{w}\|} \\ &= \frac{1}{\|\mathbf{w}\|} \left(\min_{\{\mathbf{x}_i : y_i = 1\}} |\mathbf{w} \cdot \mathbf{x}_i + b| + \min_{\{\mathbf{x}_j : y_j = -1\}} |\mathbf{w} \cdot \mathbf{x}_j + b| \right) \\ &= \frac{2}{\|\mathbf{w}\|} \end{aligned} \tag{6}$$

Hence the hyperplane that optimally separates the data is the one that minimises

$$\Phi(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 . \tag{7}$$

It is independent of *b* because provided Equation (4) is satisfied (i.e. it is a separating hyperplane) changing *b* will move it in the normal direction to itself. Accordingly the margin remains unchanged but the hyperplane is no longer optimal in that it will be nearer to one class than the other.

To consider how minimising Equation (7) is equivalent to implementing the *SRM* principle, suppose that the following bound holds,

$$\|\mathbf{w}\| \le A . \tag{8}$$

Then from Equation (4) and (5),

$$d(\mathbf{w}, b; \mathbf{x}) \ge \frac{1}{A} . \tag{9}$$

Accordingly the hyperplanes cannot be nearer than 1/A to any of the data points and intuitively it can be seen in Figure 3 how this reduces the possible hyperplanes, and hence the capacity.

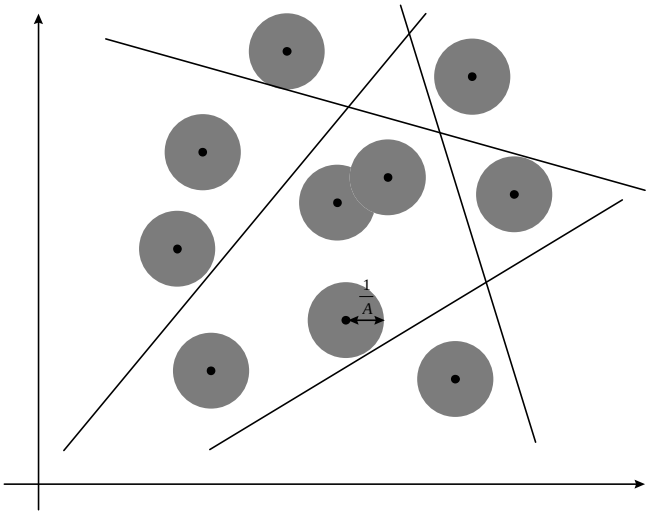


Figure 3 *Constraining the Canonical Hyperplanes*

The VC dimension, *h*, of the set of canonical hyperplanes in *n* dimensional space is,

$$h \le \min\left[R^2 A^2, n\right] + 1 , \tag{10}$$

where *R* is the radius of a hypersphere enclosing all the data points. Hence minimising Equation (7) is equivalent to minimising an upper bound on the VC dimension.

The solution to the optimisation problem of Equation (7) under the constraints of Equation (4) is given by the saddle point of the Lagrange functional (Lagrangian) [10],

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{l} \alpha_i \left\{ \left[(\mathbf{x}_i \cdot \mathbf{w}) + b \right] y_i - 1 \right\} . \tag{11}$$

where α_i are the Lagrange multipliers. The Lagrangian has to be minimised with respect to **w**, *b* and maximised with respect to $\alpha_i \geq 0$. Classical Lagrangian duality enables the *primal* problem, Equation (11), to be transformed to its *dual* problem, which is easier to solve. The *dual* problem is given by,

$$\max_{\alpha} W(\alpha) = \max_{\alpha} \left\{ \min_{\mathbf{w},b} L(\mathbf{w},b,\alpha) \right\} \tag{12}$$

The minimum with respect to **w** and *b* of the Lagrangian, *L*, is given by,

$$\begin{aligned} \frac{\partial L}{\partial b} = 0 \qquad &\Rightarrow \qquad \sum_{i=1}^{l} \alpha_i y_i = 0 \\ \frac{\partial L}{\partial \mathbf{w}} = 0 \qquad &\Rightarrow \qquad \mathbf{w} = \sum_{i=1}^{l} \alpha_i \mathbf{x}_i y_i \end{aligned} \quad . \tag{13}$$

Hence from Equations (11), (12) and (13), the *dual* problem is,

$$\max_{\alpha} W(\alpha) = \max_{\alpha} -\frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i \cdot \mathbf{x}_j \right) + \sum_{i=1}^{l} \alpha_i \tag{14}$$

with constraints,

$$\begin{aligned} \alpha_i \geq 0, \quad i = 1, \ldots, l \\ \sum_{i=1}^{l} \alpha_i y_i = 0 \end{aligned} \quad . \tag{15}$$

Solving Equation (14) with constraints Equation (15) determines the Lagrange multipliers, $\overline{\alpha}$, and the optimal separating hyperplane is given by,

$$\begin{aligned} \overline{\mathbf{w}} &= \sum_{i=1}^{l} \overline{\alpha}_i \mathbf{x}_i y_i \\ \overline{b} &= -\frac{1}{2} \overline{\mathbf{w}} \cdot \left[\mathbf{x}_r + \mathbf{x}_s \right] \end{aligned} \tag{51}$$

where $\mathbf{x}_r$ and $\mathbf{x}_s$ are any support vector from each class satisfying,

$$\overline{\alpha}_r, \overline{\alpha}_s > 0, \quad y_r = 1, \quad y_s = -1. \tag{17}$$

The classifier is then,

$$f(\mathbf{x}) = \text{sign}\left(\overline{\mathbf{w}} \cdot \mathbf{x} + \overline{b} \right) \tag{18}$$

From the Kuhn-Tucker conditions,

$$\overline{\alpha}_i \left[y_i \left(\overline{\mathbf{w}} \cdot \mathbf{x}_i + \overline{b} \right) - 1 \right] = 0 \tag{19}$$

and hence only for the points $\mathbf{x}_i$ which satisfy,

$$y_i \left(\overline{\mathbf{w}} \cdot \mathbf{x}_i + \overline{b} \right) = 1, \tag{20}$$

will the Lagrange multipliers be non-zero. These points are termed *Support Vectors* (*SV*). If the data is linearly separable all the support vectors will lie on the margin and hence the number of *SV* is typically very small. Consequently the hyperplane is

determined by a small subset of the training set; the other points could be removed from the training set and recalculating the hyperplane would produce the same answer. Hence *SVM* can be used to summarise the information contained in a data set by the *SV* produced. Points of interest,

$$\|\overline{\mathbf{w}}\|^2 = \sum_{i=1}^{l} \overline{\alpha}_i = \sum_{\text{SVs}} \overline{\alpha}_i = \sum_{\text{SVs}} \sum_{\text{SVs}} \overline{\alpha}_i \overline{\alpha}_j \left(\mathbf{x}_i \cdot \mathbf{x}_j\right) y_i y_j$$

Hence from Equation (10) the VC dimension of the classifier is bounded by,

$$h \leq \min\left[R^2 \sum_{\text{SVs}} \overline{\alpha}_i, n\right] + 1, \tag{21}$$

and if the training data, **x**, is normalised to lie in the interval $[-1,1]^n$,

$$h \leq 1 + n \min\left[\sum_{\text{SVs}} \overline{\alpha}_i, 1\right], \tag{22}$$

2.1.1 Linearly Separable Example

X_1	X_2	Class
1	1	-1
3	3	1
1	3	1
3	1	-1
2	2.5	1
3	2.5	-1
4	3	-1

Table 1 *Linearly Separable Classification Data*

Given the training set in Table 1, the *SVC* solution is shown in Figure 4. The dotted lines describe the locus of the margin and the circled data points represent the *SV*, which all lie on this margin.

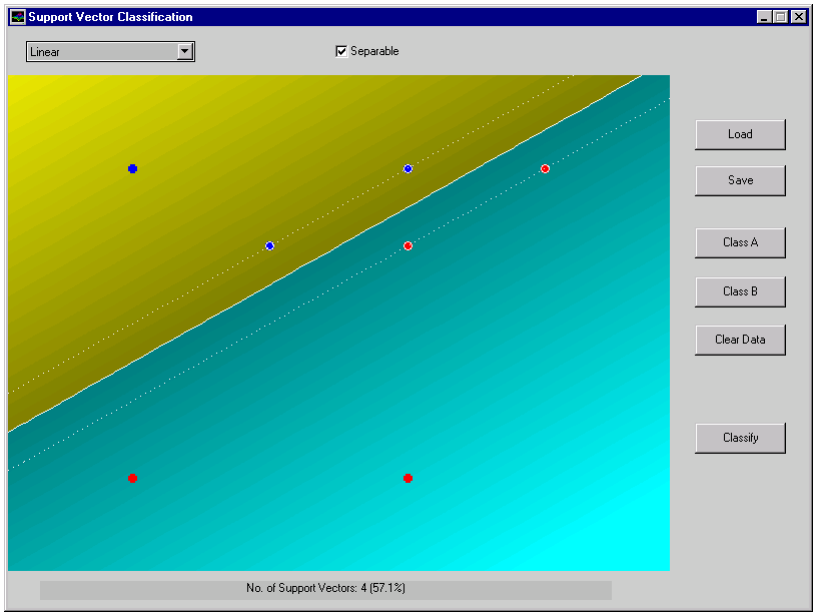


Figure 4 *Optimal Separating Hyperplane*

2.2 The Generalised Optimal Separating Hyperplane

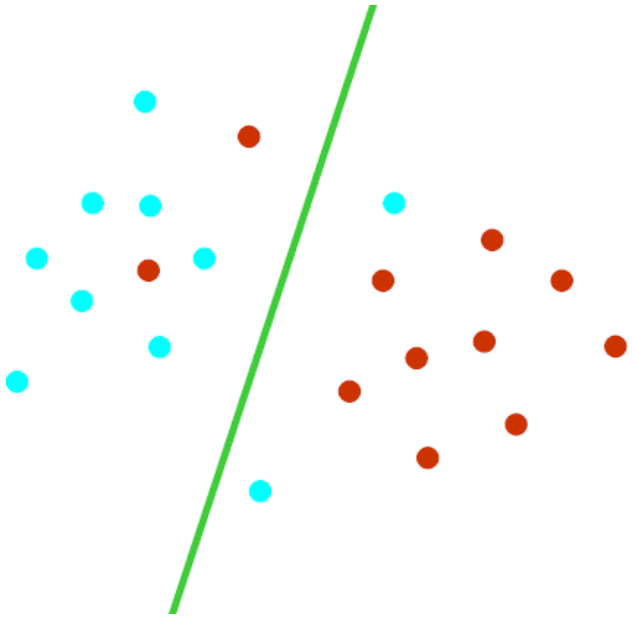


Figure 5 *Generalised Optimal Separating Hyperplane*

So far the discussion has been restricted to the case where the training data is linearly separable. However, in general this will not be the case, Figure 5. There are two approaches to generalising the problem, which are dependent upon prior knowledge of the problem and an estimate of the noise on the data. In the case where it is expected (or possibly even known) that a hyperplane can correctly separate the data, a method of introducing an additional cost function associated with misclassification is appropriate. To enable the optimal separating hyperplane method to be generalised, Cortes [5] introduced non-negative variables $\xi_i \geq 0$ and a penalty function,

$$F_\sigma(\xi) = \sum_{i=1}^{l} \xi_i^\sigma, \quad \sigma > 0,$$

where the ξ are a measure of the misclassification error. The optimisation problem is now posed so as to minimise the classification error as well as minimising the VC dimension of the classifier.

The constraints of Equation (4) are modified for the non-separable case to,

$$y_i\left[\left(\mathbf{w}\cdot\mathbf{x}_i\right)+b\right] \geq 1-\xi_i, \quad i=1,\ldots,l \tag{23}$$

where $\xi_i \geq 0$. The generalised optimal separating hyperplane is determined by the vector **w**, that minimises the functional,

$$\Phi(\mathbf{w},\xi) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i, \tag{24}$$

(where *C* is a given value) subject to the constraints of Equation (23).

The solution to the optimisation problem of Equation (24) under the constraints of Equation (23) is given by the saddle point of the Lagrangian [10],

$$L(\mathbf{w},b,\alpha) = \frac{1}{2}(\mathbf{w}\cdot\mathbf{w}) + C\sum_{i=1}^{l}\xi_i - \sum_{i=1}^{l}\alpha_i\left\{\left[\left(\mathbf{x}_i\cdot\mathbf{w}\right)+b\right]y_i - 1 + \xi_i\right\} - \sum_{i=1}^{l}\beta_i\xi_i, \tag{25}$$

where α_i, β_i are the Lagrange multipliers. The Lagrangian has to be minimised with respect to **w**, *b*, ξ and maximised with respect to $\alpha_i, \beta_i \geq 0$. As before, classical Lagrangian duality enables the *primal* problem, Equation (25), to be transformed to its *dual* problem. The *dual* problem is given by,

$$\max_{\alpha,\beta} W(\alpha) = \max_{\alpha,\beta}\left\{\min_{\mathbf{w},b,\xi} L(\mathbf{w},b,\xi,\alpha,\beta)\right\} \tag{26}$$

The minimum with respect to **w**, *b* and ξ_i of the Lagrangian, *L*, is given by,

$$\begin{aligned}
\frac{\partial L}{\partial b} = 0 \quad &\Rightarrow \quad \sum_{i=1}^{l}\alpha_i y_i = 0 \\
\frac{\partial L}{\partial \mathbf{w}} = 0 \quad &\Rightarrow \quad \mathbf{w} = \sum_{i=1}^{l}\alpha_i\mathbf{x}_i y_i . \\
\frac{\partial L}{\partial \xi_i} = 0 \quad &\Rightarrow \quad \alpha_i + \beta_i = C
\end{aligned} \tag{27}$$

Hence from Equations (25), (26) and (27), the *dual* problem is,

$$\max_{\alpha} W(\alpha) = \max_{\alpha} -\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j\left(\mathbf{x}_i\cdot\mathbf{x}_j\right) + \sum_{i=1}^{l}\alpha_i \tag{28}$$

with constraints,

$$0 \le \alpha_i \le C, \quad i = 1,\ldots,l$$
$$\sum_{i=1}^{l} \alpha_i y_i = 0 \quad . \tag{29}$$

The solution to this minimisation problem is identical to the separable case except for a modification of the bounds of the Lagrange multipliers. The uncertain part of Cortes's approach is that the coefficient *C* has to be determined. In some circumstances *C* can be directly related to a regularisation parameter [7,15]. Blanz [4] uses a value of *C*=5, but ultimately *C* must be chosen to reflect the knowledge of the noise on the data. This warrants further work, but a more practical discussion is given in Chapter 4.

2.2.1 Linearly Non-Separable Example

X_1	X_2	Class
1	1	-1
3	3	1
1	3	1
3	1	-1
2	2.5	1
3	2.5	-1
4	3	-1
1.5	1.5	1
1	2	-1

Table 2 *Linearly Non-Separable Classification Data*

Two additional data points are added to the separable data of Table 1 to produce a linearly non-separable data set, Table 2. The *SVC* is shown in Figure 6. The *SV* are no longer required to lie on the margin, as in Figure 4.

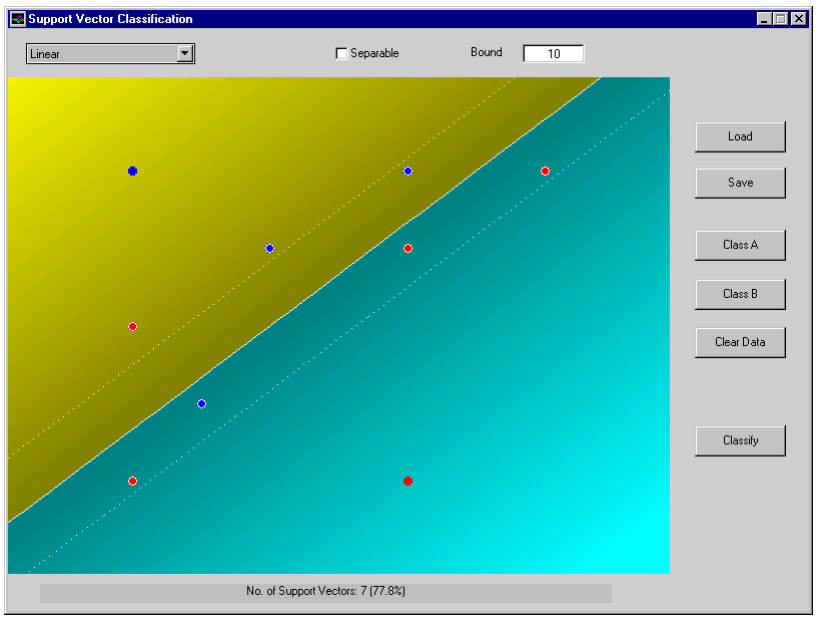


Figure 6 *Generalised Optimal Separating Hyperplane Example (C=10)*

In contrast as $C \to \infty$ the solution converges towards the solution of obtained by the optimal separating hyperplane, Figure 7.

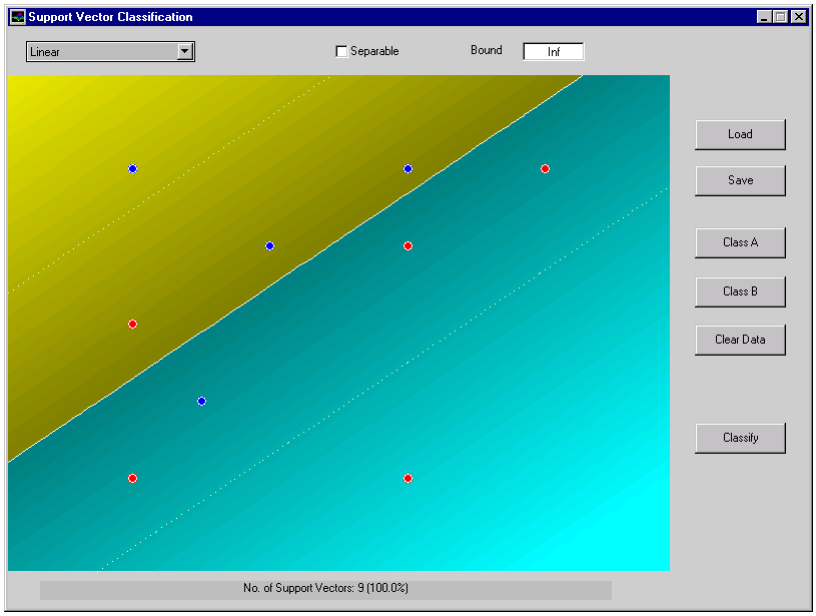


Figure 7 *Generalised Optimal Separating Hyperplane Example (C=∞)*

In the limit as $C \to 0$ the solution converges to ??, Figure 8.

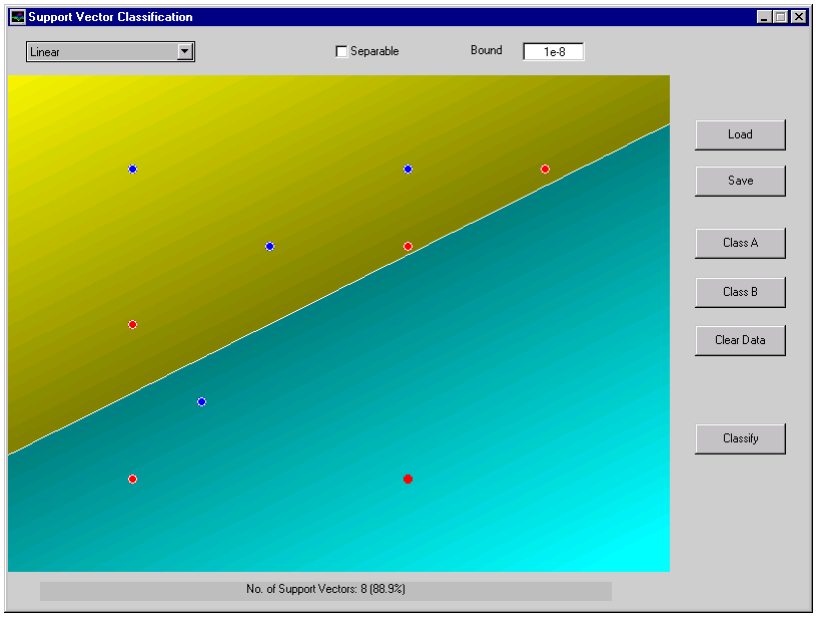


Figure 8 *Generalised Optimal Separating Hyperplane Example ($C=10^{-8}$)*

2.3 Generalisation in High Dimensional Feature Space

In the case where a linear boundary is inappropriate the *SVM* can map the input vector, **x**, into a high dimensional feature space, **z**. By choosing a non-linear mapping *a priori*, the *SVM* constructs an optimal separating hyperplane in this higher dimensional space, Figure 9. The idea exploits the method of [1] which enables the curse of dimensionality [3] to be addressed.

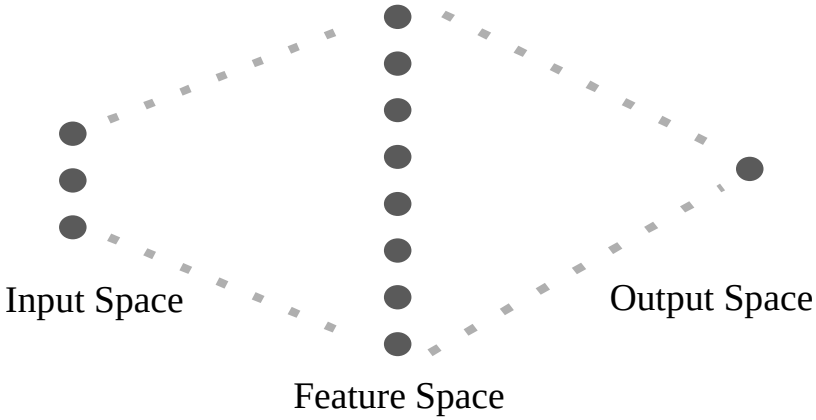


Figure 9 *Mapping the Input Space into a High Dimensional Feature Space*

There are some restrictions on the of non-linear mapping that can be employed, see Chapter 3, but it turns out that most commonly employed functions are acceptable. Among the acceptable mappings are polynomials, radial basis functions and certain sigmoid functions.

The optimisation problem of Equation (28) becomes,

$$\max_{\alpha} W(\alpha) = \max_{\alpha} -\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j K(\mathbf{x}_i,\mathbf{x}_j) + \sum_{i=1}^{l}\alpha_i \tag{30}$$

where $K(\mathbf{x},\mathbf{y})$ is the kernel function performing the non-linear mapping into feature space, and the constraints are unchanged,

$$\begin{aligned} &\alpha_i \geq 0, \quad i = 1,\ldots,l \\ &\sum_{i=1}^{l}\alpha_i y_i = 0 \end{aligned} \quad . \tag{31}$$

Solving Equation (30) with constraints Equation (31) determines the Lagrange multipliers, $\bar{\alpha}$, and the classifier implementing the optimal separating hyperplane in the feature space is given by,

$$f(\mathbf{x}) = \text{sign}\left(\sum_{\text{SVs}}\bar{\alpha}_i y_i K(\mathbf{x}_i,\mathbf{x}) + \bar{b}\right) \tag{57}$$

where

$$\bar{b} = -\frac{1}{2}\sum_{\text{SVs}}\bar{\alpha}_i y_i\left[K(\mathbf{x}_r,\mathbf{x}_i) + K(\mathbf{x}_s,\mathbf{x}_i)\right]. \tag{33}$$

(Alternatively a more stable way of computing b can be done [18])

If the Kernel contains a bias term, b can be accommodated within the Kernel function, and hence the classifier is simply,

$$f(\mathbf{x}) = \text{sign}\left(\sum_{\text{SVs}}\bar{\alpha}_i y_i K(\mathbf{x}_i,\mathbf{x})\right). \tag{57}$$

Many employed Kernels have a bias term and any finite Kernel can be made to have one [7]. Note here that provided the Kernel contains a 'bias term' the term b may be

dropped from the equation of the hyperplane, simplifying the optimisation problem by removing the equality constraint of Equation (31). Chapter 3 discusses in more detail the choice of Kernel functions and the conditions that are imposed.

2.3.1 Polynomial Mapping Example

Consider the Kernel of the form,

$$K(\mathbf{x},\mathbf{y}) = \left[(\mathbf{x}\cdot\mathbf{y})+1\right]^2. \tag{35}$$

Applying the non-linear *SVC* to the linearly non-separable training data of Table 2, produces the classification illustrated in Figure 10 (C=∞). The margin is no longer of constant width due to the non-linear projection into the input space. The solution is in contrast to Figure 6-8, in that the training data is now classified correctly. However, even though *SVM* implement the *SRM* principle and hence they should generalise well, careful choice of the kernel function is necessary to produce a classification boundary that is topologically appropriate. It is always possible to map the input space into a dimension greater than the number of training points and produce a 'perfect' classifier on the training set. However, this will generalise badly. The choice of kernel warrants further investigation.

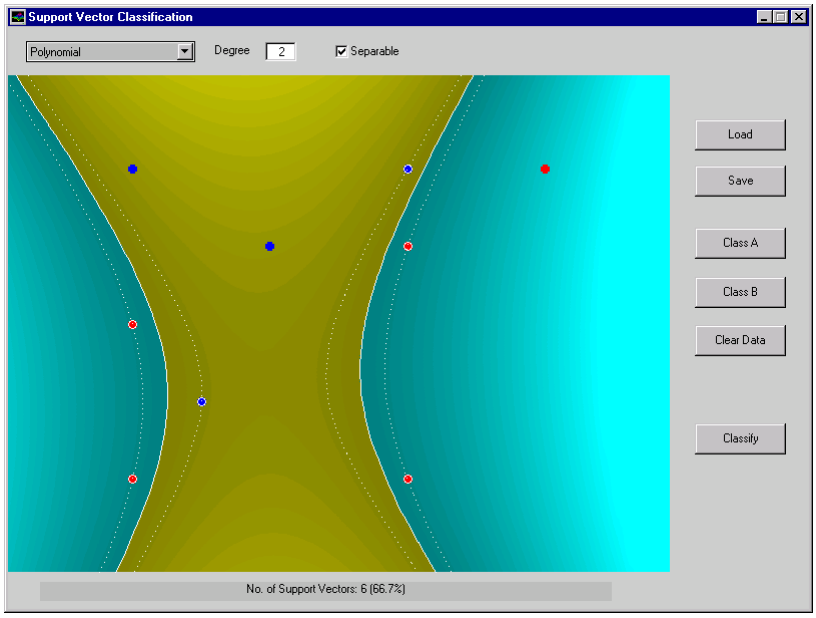


Figure 10 *Mapping input space into Polynomial Feature Space*

2.4 Discussion

Typically the data will only be linearly separable in some, possibly very high dimensional feature space. It may not make sense to try and separate the data exactly, particularly when only a finite amount of training data which is potentially corrupted by some kind of noise. Hence in practice it will be necessary to employ the non-separable approach which places an upper bound on the Lagrange multipliers. This raises the question of how to determine the parameter C. It is similar to the problem in

regularisation where the regularisation coefficient has to be determined. Here the parameter can be determined by a process of cross-validation, and it may be possible to implement this here. Note that removing the training patterns that are not support vectors will not change the solution and hence a fast method may be available when the support vectors are sparse.

3 Feature Space

3.1 Kernel Functions

The following theory is based upon Reproducing Kernel Hilbert Spaces (*RKHS*) [2, 20, 7, 9]. The idea of the kernel function is to perform the operations in the input space rather than the potentially high dimensional feature space. Hence the inner product does not need to be evaluated in the feature space. A inner product in feature space has an equivalent kernel in input space,

$$K(\mathbf{x},\mathbf{y}) = k(\mathbf{x}) \cdot k(\mathbf{y}), \tag{36}$$

provide certain conditions hold. This is appropriate in our case if K is a symmetric positive definite function, which satisfies Mercer's Conditions,

$$\begin{gathered} K(\mathbf{x},\mathbf{y}) = \sum_{m=1}^{\infty} \alpha_m \psi(\mathbf{x}) \psi(\mathbf{y}), \qquad \alpha_m \geq 0 \\ \iint K(\mathbf{x},\mathbf{y}) g(\mathbf{x}) g(\mathbf{y}) d\mathbf{x} d\mathbf{y} > 0, \quad \int g^2(\mathbf{x}) d\mathbf{x} < \infty \end{gathered} \tag{37}$$

Popular functions which satisfy Mercer's conditions are,

3.1.1 Polynomial

$$\begin{gathered} K(\mathbf{x},\mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d \\ ,d = 1,... \\ K(\mathbf{x},\mathbf{y}) = ((\mathbf{x} \cdot \mathbf{y}) + 1)^d \end{gathered} \tag{38}$$

The second is preferable as it avoids problems with the hessian becoming zero.

3.1.2 Gaussian Radial Basis Function

$$K(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{(\mathbf{x}-\mathbf{y})^2}{2\sigma^2} \right) \tag{39}$$

Classical techniques utilising radial basis functions employ some method of determining a subset of centres. This selection is implicit when employed within an *SVM*. This local function is attractive in the sense that the non-zero support vectors each contribute one local Gaussian function, centred at that data point. By further considerations it is possible to select the global basis function width, σ, using the *SRM* principle [19].

3.1.3 Exponential Radial Basis Function

$$K(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{|\mathbf{x}-\mathbf{y}|}{2\sigma^2} \right) \tag{40}$$

This function has s

3.1.4 Multi-Layer Perceptron

$$K(\mathbf{x},\mathbf{y}) = \tanh\left(b(\mathbf{x}\cdot\mathbf{y}) - c\right) \tag{41}$$

For some values of b and c.

3.1.5 Fourier Series

Fourier series can be considered an expansion in the following $2N+1$ dimensional feature space. The kernel is defined over the interval

$$K(\mathbf{x},\mathbf{y}) = \frac{\sin\left(N+\frac{1}{2}\right)(\mathbf{x}-\mathbf{y})}{\sin\left(\frac{1}{2}(\mathbf{x}-\mathbf{y})\right)} \tag{42}$$

3.1.6 Linear Splines

It is also possible to use infinite spline kernels,

$$K(x,y) = 1 + xy + xy\min(x,y) - \frac{(x+y)}{2}\left(\min(x,y)\right)^2 + \frac{1}{3}\left(\max(x,y)\right)^3 \tag{43}$$

This kernel is defined on the interval [0,1) ([0,inf))

3.1.7 B_n splines

The kernel is defined on the interval [-1,1]

$$K(x,y) = B_{2n+1}(x-y) \tag{44}$$

3.1.8 Tensor Product Splines

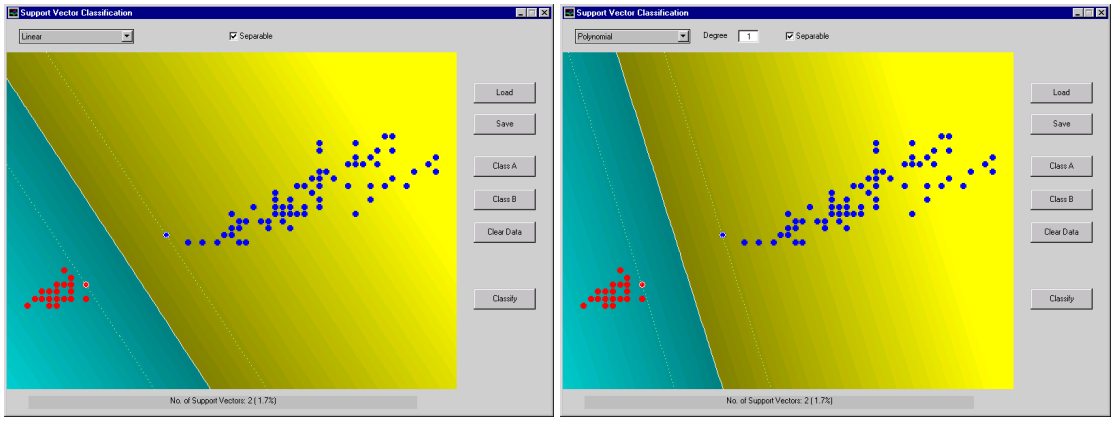
Multidimensional spline kernels can be obtained by forming tensor products,

$$K(\mathbf{x},\mathbf{y}) = \prod_{m=1}^{n} K_m(\mathbf{x}_m,\mathbf{y}_m) \tag{45}$$

The

3.2 Implicit vs. Explicit Bias

The solutions with an implicit bias and explicit bias are not the same, which may initially come as a surprise. However, the difference helps to highlight the problem with the interpretation of generalisation in high dimensional feature spaces.



(a) Explicit (linear) (b) Implicit (polynomial degree 1)

Figure 11 *Comparison between Implicit and Explicit bias for a linear kernel.*

Comparison of linear with explicit bias against polynomial of degree 1 with implicit bias.

3.3 Data Normalisation

If the data is not normalised there may be too much emphasis on one input. If it is normalised this will effect the solution. Is this effect predictable? Also some kernels are only valid over a restricted interval and as such demand data normalisation. Empirical Observations suggest that Normalisation also improves the condition number of the matrix in the optimisation problem.

4 Classification Example: IRIS data

The iris data set is a well known data set used for demonstrating the performance of classification algorithms. The data set contains four attributes of an iris, and the goal is to classify the class of iris based on these four attributes. To simplify the problem we restrict ourselves to the two features which contain the most information about the class, namely the petal length and the petal width. The distribution of the data is illustrated in Figure 12.

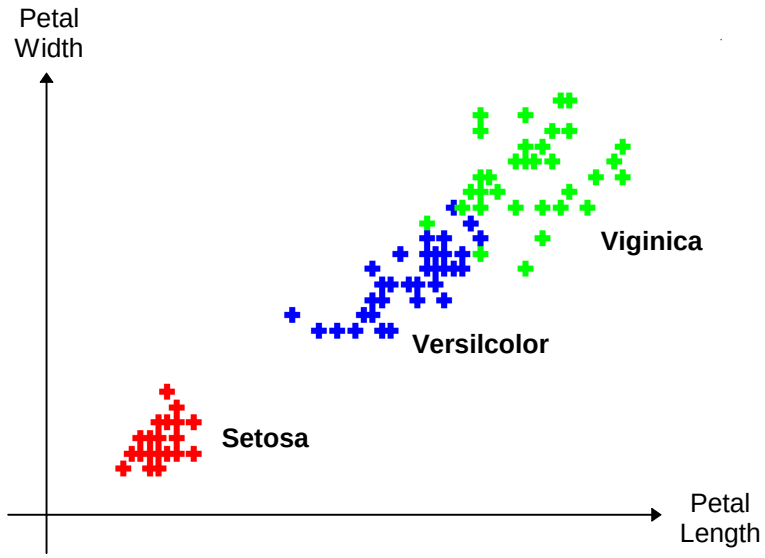


Figure 12 *Iris data set*

This data set has been widely used and [17] has applied *SVM* to it, which provides a means of verifying the operation of the software. The Setosa and Versilcolor classes are easily separated with a linear boundary and the support vector solution using an inner product kernel is illustrated in Figure 13.

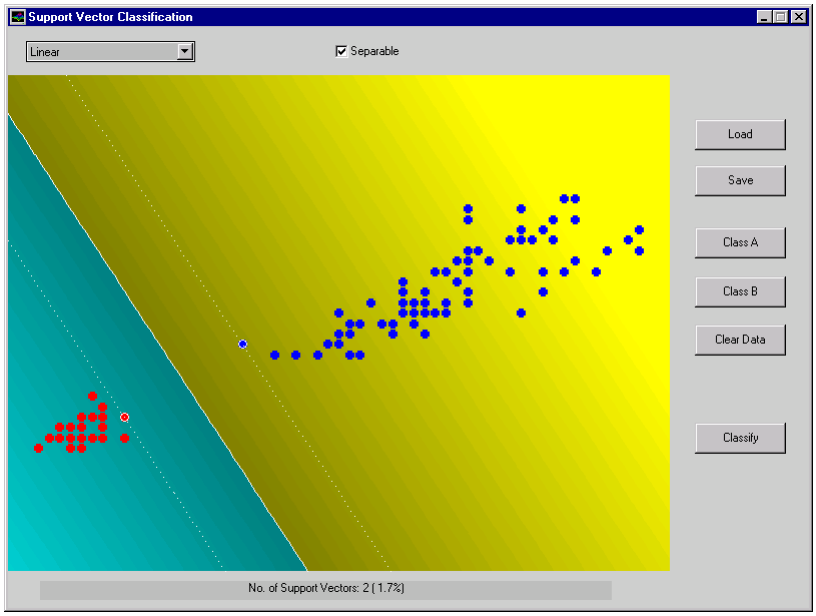


Figure 13 *Separating Setosa with a linear SVM*

Here the support vectors are circled. It is evident that there are only two support vectors. These vectors contain the important information about the classification boundary and hence suggest that the support vector machine can be used to extract the training patterns that contain the most information for the classification problem. The separation of the class Viginica from the other two classes is not so trivial. In fact, two of the examples are identical in petal length and width, but correspond to different classes.

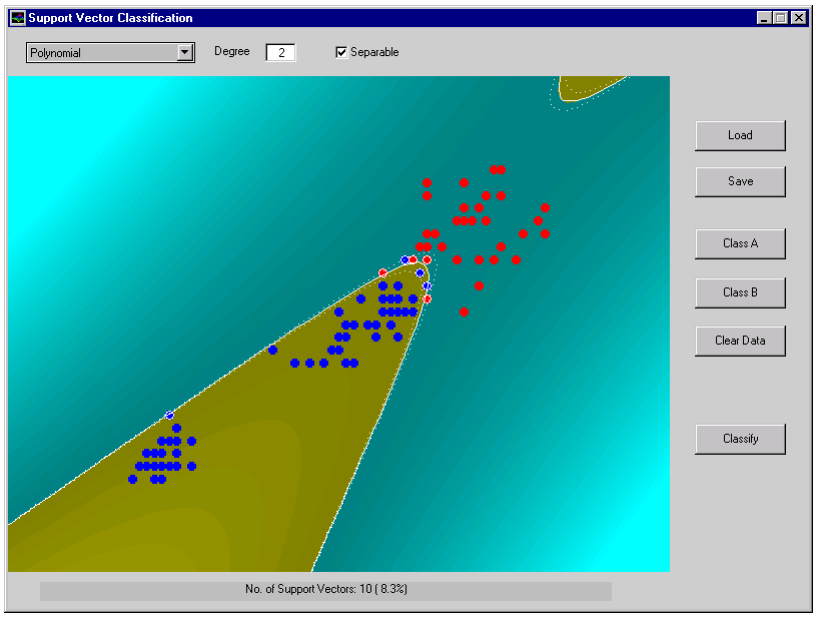


Figure 14 *Separating Viginica with a polynomial SVM (degree 2)*

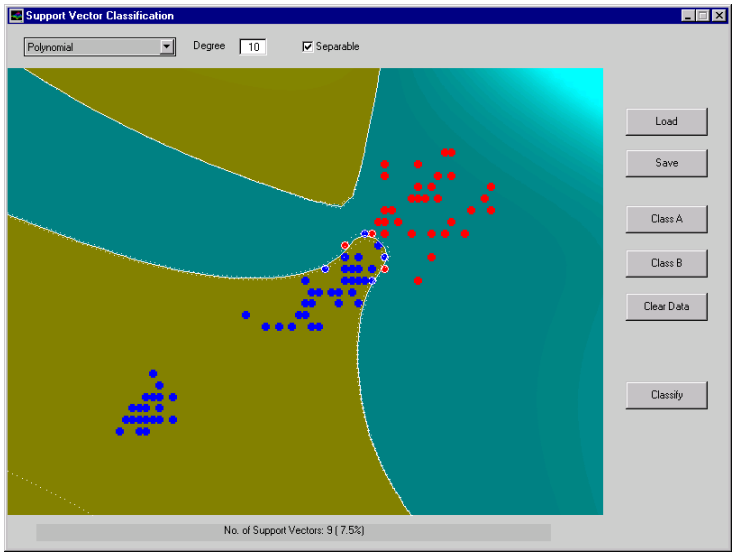


Figure 15 *Separating Viginica with a polynomial SVM (degree 10)*

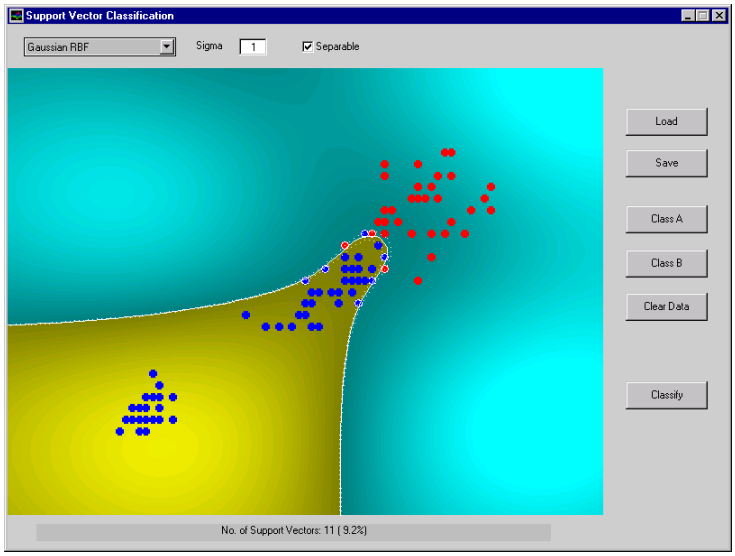


Figure 16 *Separating Viginica with a Radial Basis Function SVM (σ=1.0)*

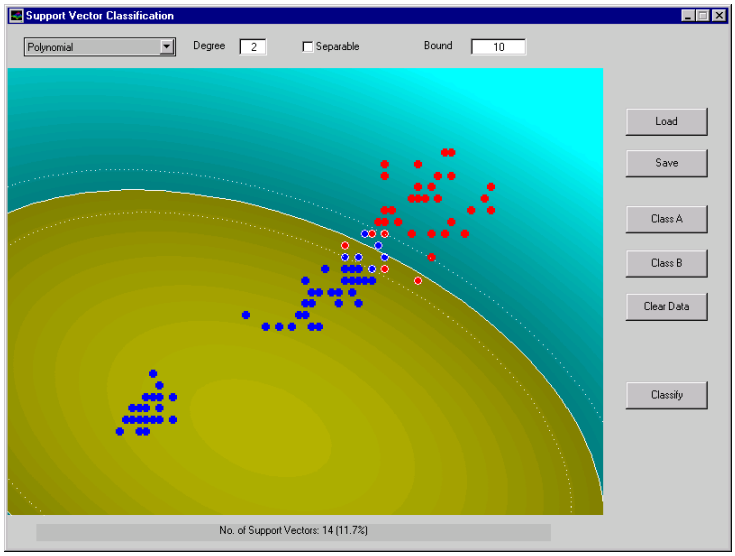
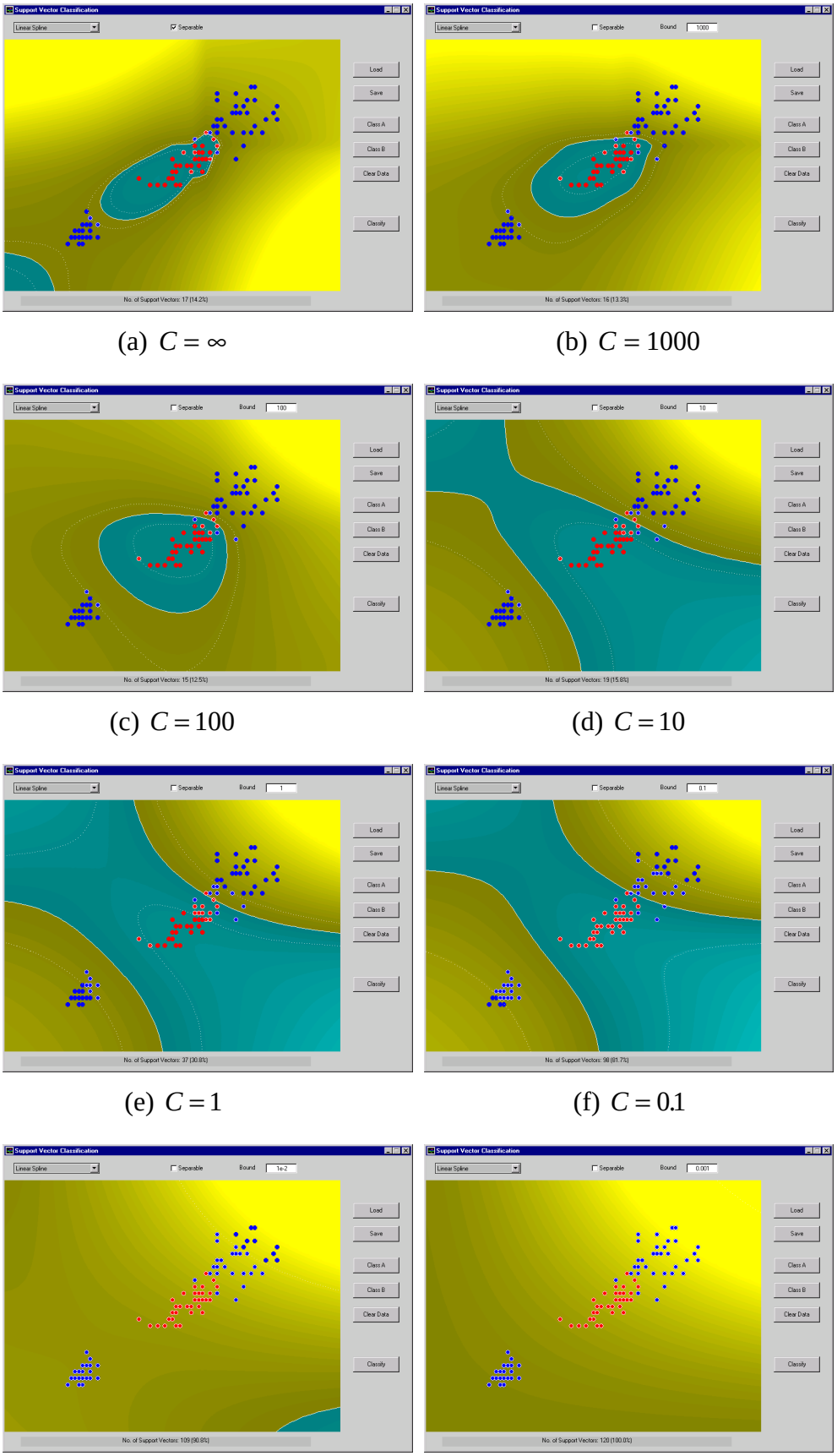


Figure 17 *Separating Viginica with a polynomial SVM (degree 2, C=10)*



(a) $C = \infty$ (b) $C = 1000$

(c) $C = 100$ (d) $C = 10$

(e) $C = 1$ (f) $C = 0.1$

(g) $C = 0.01$ (h) $C = 0.001$

Figure 18 *The effect of C on the separation of Versilcolor with a linear spline SVM*

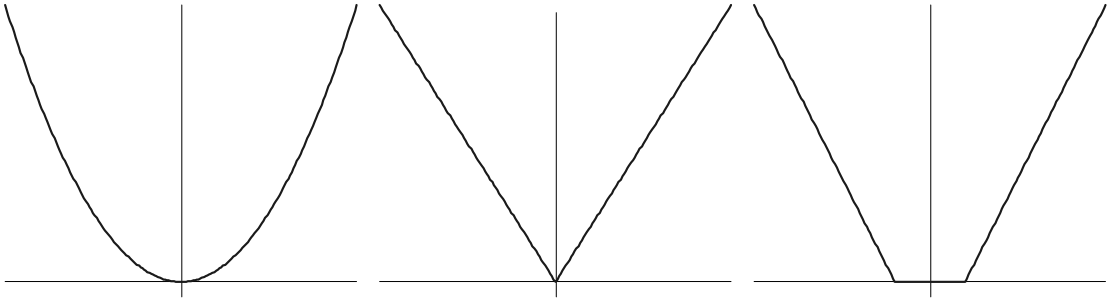
Figure 18 illustrates how the classification boundary changes with the parameter *C* which controls the tolerance to misclassification errors. Interestingly, the range of values [0.1,1000] provide sensible boundaries, but to know whether an open boundary (e.g. Figure 18(e)) or a closed boundary (e.g. Figure 18(c)) is more appropriate would require prior knowledge about the problem under consideration.

It would be expected that Figure 13 and Figure 17 would give reasonable generalisation.

[13] applies *SVC* to face recognition.

5 Support Vector Regression

SVM can be applied to regression problems. The main difference is the type of a loss function employed. Figure 19 illustrates three possible loss functions



(a) Quadratic (b) Least Modulus (c) ε-Insensitive

Figure 19 *Loss Functions*

Why is this so important? because it makes the SVs sparse. Robust regression in the sense of Huber. Data is always non-separable? [0,inf] [-Inf,0] class [0,e] [-e,-0] regress

(a) is equivalent to standard least squares error criterion.

5.1 Linear Regression

Consider the problem of approximating the set of training vectors,

$$(y_1, \mathbf{x}_1), \ldots, (y_l, \mathbf{x}_l), x \in R^n, y \in R \tag{46}$$

with a linear function,

$$f(\mathbf{x}) = (\mathbf{w} \cdot \mathbf{x}) + b \tag{47}$$

minimise the functional,

$$\Phi(\mathbf{w}, \xi^*, \xi) = \frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) + C\left(\sum_{i=1}^{l} \xi_i + \sum_{i=1}^{l} \xi_i^*\right), \tag{48}$$

(where C is a given value). The solution is given by,

$$\max_{\alpha,\alpha^*} W(\alpha, \alpha^*) = \max_{\alpha,\alpha^*} \left\{ \begin{array}{c} \sum_{i=1}^{l} \alpha_i^* (y_i - \varepsilon) - \alpha_i (y_i + \varepsilon) \\ -\frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)(\mathbf{x}_i \cdot \mathbf{x}_j) \end{array} \right\} \tag{49}$$

with constraints,

$$0 \le \alpha_i \le C, \quad i = 1, \ldots, l$$

$$0 \le \alpha_i^* \le C, \quad i = 1, \ldots, l. \qquad (50)$$

$$\sum_{i=1}^{l} (\alpha_i^* - \alpha_i) = 0$$

Solving Equation (49) with constraints Equation (50) determines the Lagrange multipliers, $\bar{\alpha}, \bar{\alpha}^*$, and the regression function is given by,

where

$$\bar{\mathbf{w}} = \sum_{i=1}^{l} (\bar{\alpha}_i - \bar{\alpha}_i^*) \mathbf{x}_i$$

$$\bar{b} = -\frac{1}{2} \bar{\mathbf{w}} \cdot [\mathbf{x}_r + \mathbf{x}_s] \qquad (51)$$

(Alternatively a more stable way of computing b can be done [smola])

$$f(\mathbf{x}) = \bar{\mathbf{w}} \cdot \mathbf{x} + \bar{b} \qquad (55)$$

It can be shown that,

$$\bar{\alpha}_i \bar{\alpha}_i^* = 0, \quad i = 1, \ldots, l.$$

Therefore the support vectors are points where exactly one of the Lagrange multipliers is greater than zero.

5.1.1 Example

Given the following training set,

X	Y
1.0	-1.6
3.0	-1.8
4.0	-1.0
5.6	1.2
7.8	2.2
10.2	6.8
11.0	10.0
11.5	10.0
12.7	10.0

Table 3 *Regression Data*

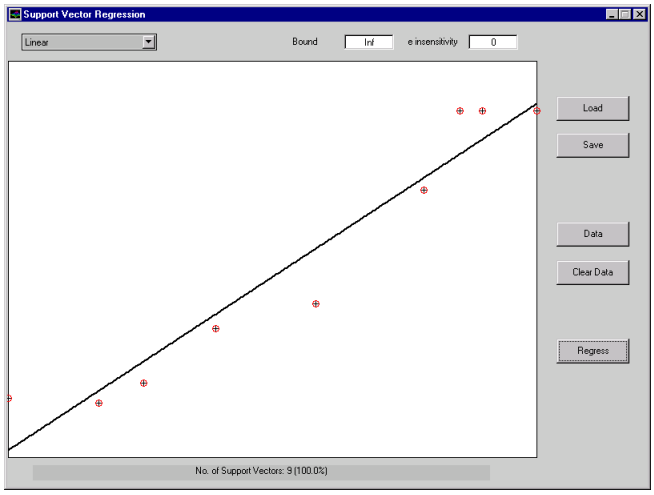


Figure 20 *Linear regression*

5.2 Non Linear Regression

The support vector machine maps the input vector, **x**, into a high dimensional feature space, **z**, through some non-linear mapping, chosen *a priori*. In this space an optimal separating hyperplane is constructed.

(where C is a given value). The solution is given by,

$$\max_{\alpha,\alpha^*} W(\alpha,\alpha^*) = \max_{\alpha,\alpha^*} \left\{ \begin{array}{c} \sum_{i=1}^{l} \alpha_i^* (y_i - \varepsilon) - \alpha_i (y_i + \varepsilon) \\ -\frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j) K(\mathbf{x}_i, \mathbf{x}_j) \end{array} \right\} \tag{53}$$

with constraints,

$$\begin{array}{c} 0 \le \alpha_i \le C, \quad i = 1,\ldots,l \\ 0 \le \alpha_i^* \le C, \quad i = 1,\ldots,l. \\ \sum_{i=1}^{l} (\alpha_i^* - \alpha_i) = 0 \end{array} \tag{54}$$

if the kernel function contains a bias term. Solving Equation (49) with constraints Equation (50) determines the Lagrange multipliers, $\bar{\alpha}, \bar{\alpha}^*$, and the regression function is given by,

$$f(\mathbf{x}) = \sum_{\text{SVs}} (\bar{\alpha}_i - \bar{\alpha}_i^*) K(\mathbf{x}_i, \mathbf{x}) + \bar{b} \tag{55}$$

where

$$\bar{b} = -\frac{1}{2} \sum_{\text{SVs}} (\bar{\alpha}_i - \bar{\alpha}_i^*) \left[K(\mathbf{x}_r, \mathbf{x}_i) + K(\mathbf{x}_s, \mathbf{x}_i) \right]. \tag{56}$$

(Alternatively a more stable way of computing b can be done [smola])

As with the SVC the equality constraint may be dropped if the Kernel contains a bias term, *b* being accommodated within the Kernel function, and the regression function is given by,

$$f(\mathbf{x}) = \sum_{\text{SVs}} (\overline{\alpha}_i - \overline{\alpha}_i^*) K(\mathbf{x}_i, \mathbf{x}). \tag{57}$$

5.2.1 Case 1: The Separable Case

$$W(\alpha) = \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^{l} \alpha_i \tag{58}$$

5.2.2 Polynomial Learning Machine

$$K(x, x_i) = \left[(x \cdot x_i) + 1 \right]^d \tag{59}$$

5.2.3 Example

Given the following training set,

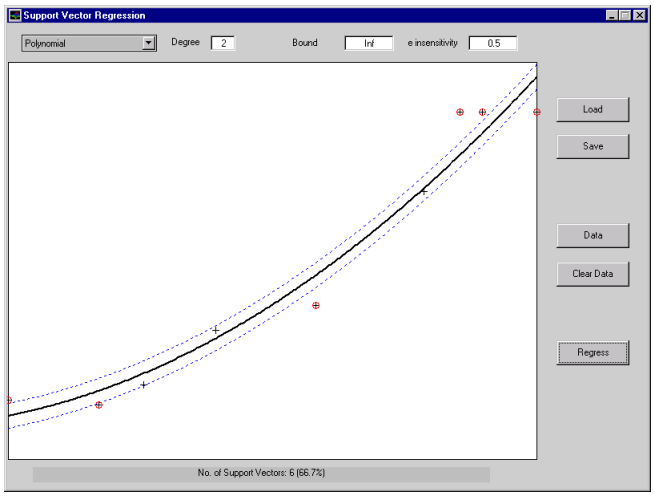


Figure 21 *Polynomial Regression*

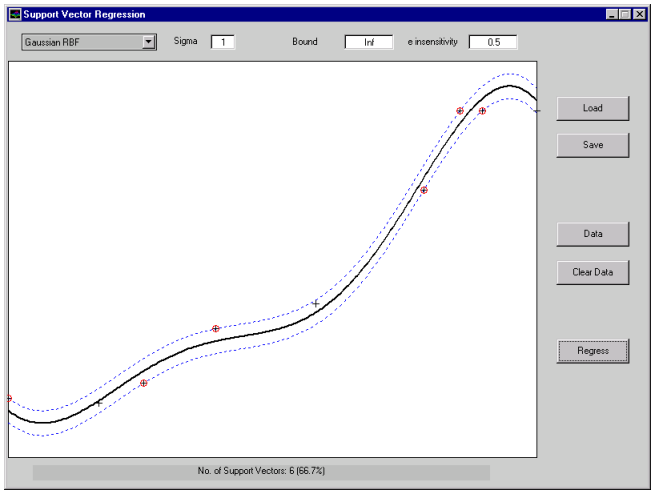


Figure 22 *Radial Basis Function Regression*

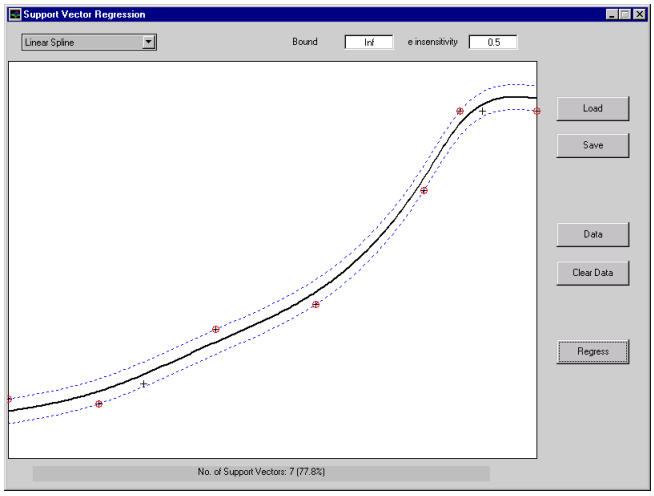


Figure 23 *Infinite Spline Regression*

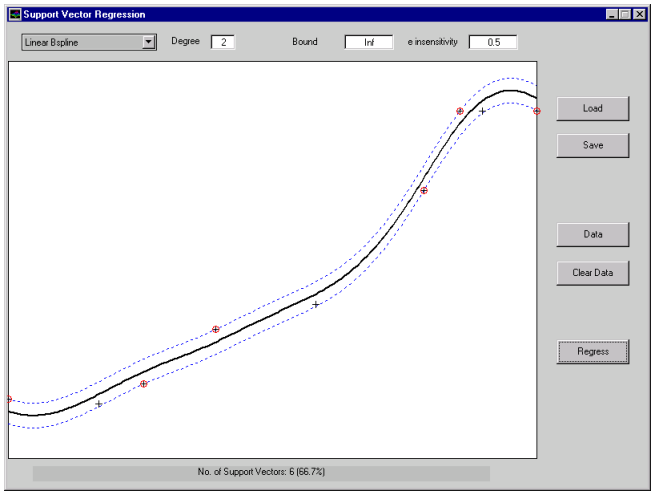


Figure 24 *Infinite B-spline regression*

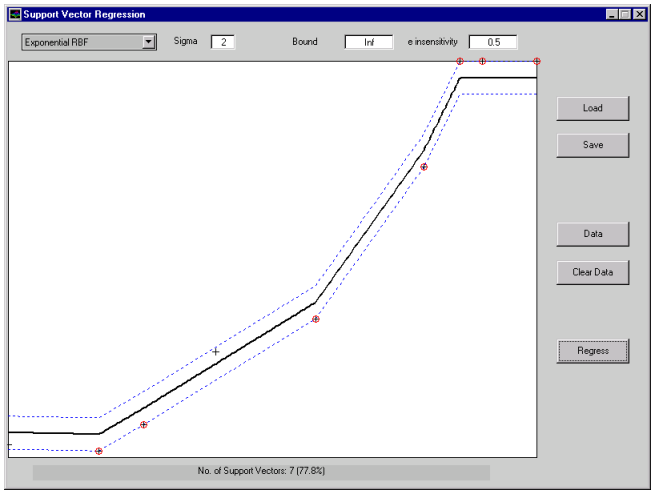


Figure 25 *Exponential RBF*

6 Regression Example: Titanium Data

[12] has achieved excellent results applying svms to time series from santa fe set ?.

[11] has applied SVMs to time series modelling

The example given here considers the titanium data [6] as an illustrative example for one dimensional non-linear regression. There two parameters to control the regression, *C* which controls the

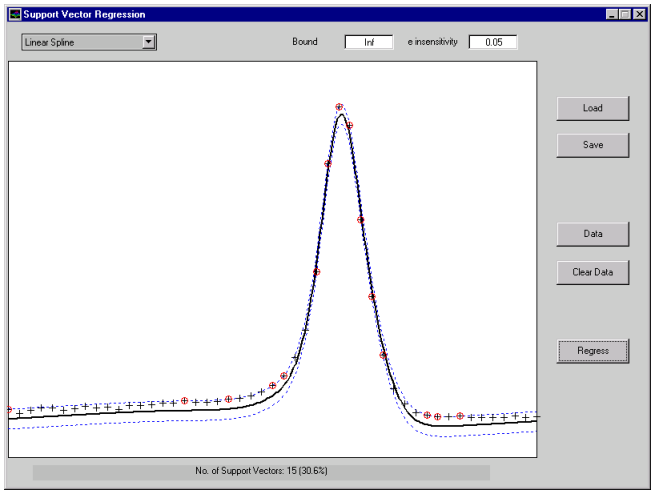


Figure 26 *Linear Spline Regression (*ε=0.05*)*

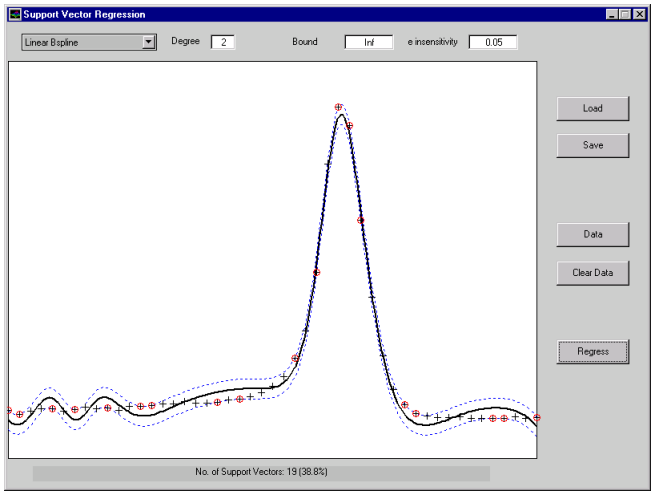


Figure 27 *B-Spline Regression (*ε=0.05*)*

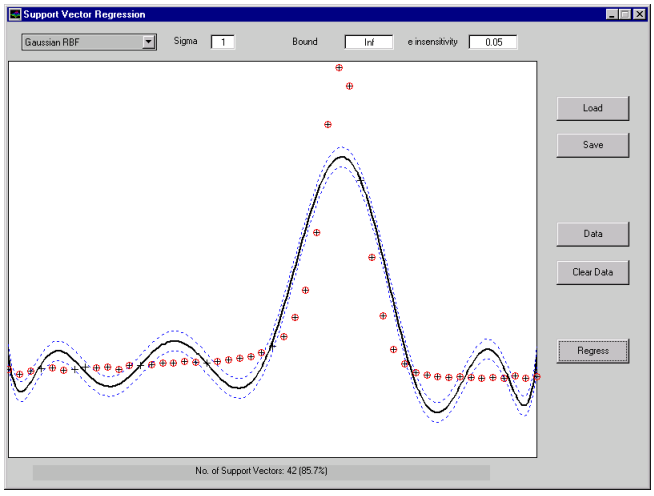


Figure 28 *Gaussian RBF Regression (*ε=0.05, σ=1.0*)*

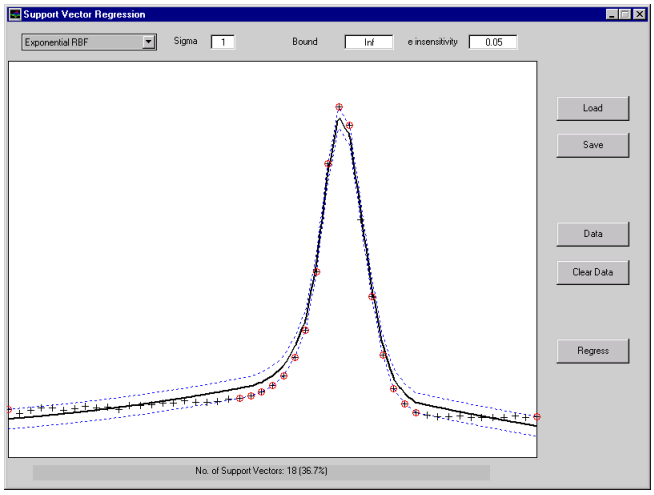


Figure 29 *Exponential RBF Regression (*ε=0.05, σ=1.0*)*

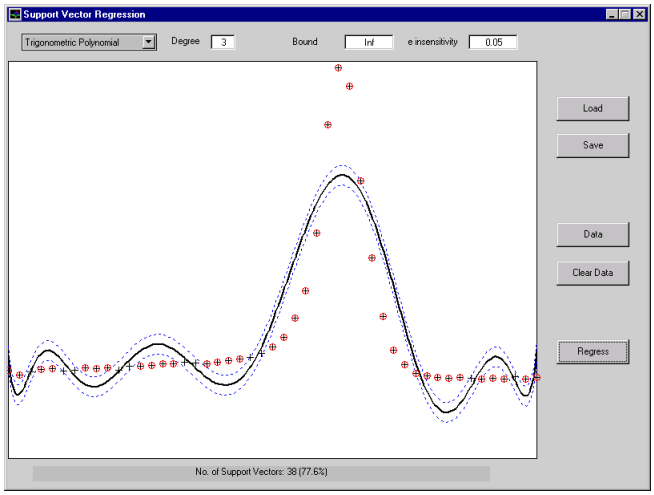


Figure 30 *Fourier Regression (*ε=0.05, degree 3*)*

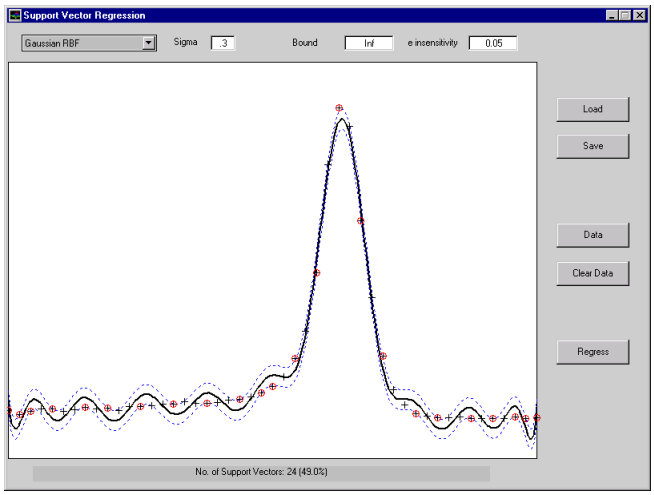


Figure 31 *Gaussian RBF Regression (*ε=0.05, σ=0.3*)*

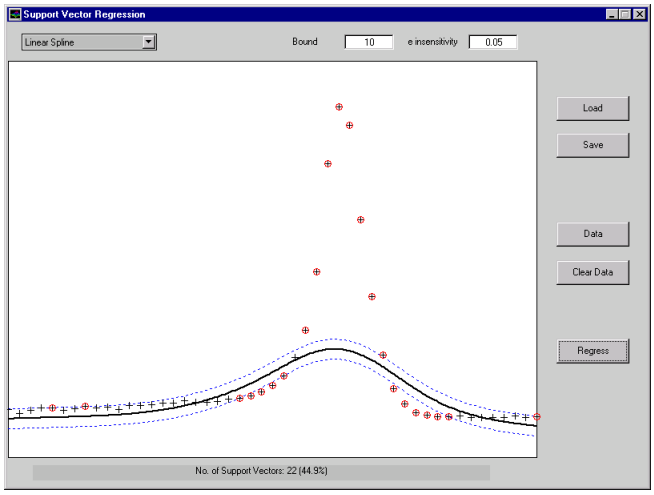


Figure 32 *Linear Spline Regression (*ε=0.05, C=10*)*

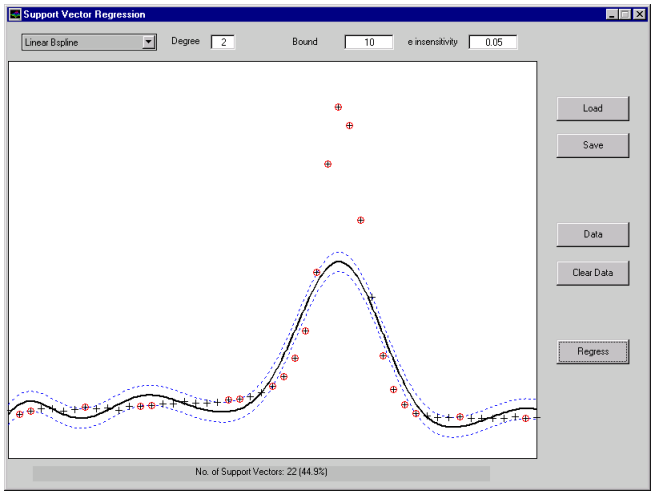


Figure 33 *B-Spline Regression (*ε=0.05, C=10*)*

7 Conclusions

Strong theoretical foundation

global minimum

quadratic programming

margin in feature space -> input space

choice of C, e

choice of kernel function, model mismatch

invariances

curse of dimensionality shift problem to training data size.

References

[1] M. Aizerman, E. Braverman and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, **25**:821-837, 1964.

[2] N. Aronszajn. Theory of Reproducing Kernels. *Trans. Amer. Math. Soc.*, **68**:337-404, 1950.

[3] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[4] V. Blanz, B. Schölkopf, H. Bülthoff, C. Burges, V. Vapnik, and T. Vetter. 1996. Comparison of view-based object recognition algorithms using realistic 3D models. In: C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff (eds.): *Artificial Neural Networks - ICANN'96*. Springer Lecture Notes in Computer Science Vol. 1112, Berlin, 251-256.

[5] C. Cortes, and V. Vapnik. 1995. Support Vector Networks. *Machine Learning* 20:273-297.

[6] P. Dierckx. *Curve and Surface Fitting with Splines*. Monographs on Numerical Analysis, Clarendon Press, Oxford, 1993.

[7] F. Girosi. An Equivalence Between Sparse Approximation and Support Vector Machines. Technical Report AIM-1606, Artificial Intelligence Laboratory, Massachusetts Institute of Technology (MIT), Cambridge, Massachusetts, 1997.

[8] S. R. Gunn, M. Brown and K. M. Bossley. Network Performance Assessment for Neurofuzzy Data Modelling. *Lecture Notes in Computer Science*, **1280**:313-323, 1997.

[9] N. E. Heckman. The Theory and Application of Penalized Least Squares Methods or Reproducing Kernel Hilbert Spaces Made Easy, 1997. ftp://newton.stat.ubc.ca/pub/nancy/PLS.ps

[10] M. Minoux. *Mathematical Programming: Theory and Algorithms*. John Wiley and Sons, 1986.

[11] S. Mukherjee, E. Osuna, and F. Girosi. Nonlinear Prediction of Chaotic Time Series using Support Vector Machines. To appear *in Proc. of IEEE NNSP'97*, Amelia Island, FL, 24-26 Sep., 1997.

[12] K. R. Mülller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen and V. Vapnik. 1997. Predicting Time Series with Support Vector Machines. Submitted to: *ICANN'97.*

[13] E. Osuna, R. Freund and F. Girosi. An Improved Training Algorithm for Support Vector Machines. To appear *in Proc. of IEEE NNSP'97*, Amelia Island, FL, 24-26 Sep., 1997.

[14] E. Osuna, R. Freund and F. Girosi. 1997. Improved Training Algorithm for Support Vector Machines. To appear in: *NNSP'97.*

[15] A. Smola and B. Schölkopf. 1997. On a Kernel-based Method for Pattern Recognition, Regression, Approximation and Operator Inversion. GMD Technical Report No. 1064. (To appear in *Algorithmica*, special issue on Machine Learning)

[16] M. O. Stitson and J. A. E. Weston. Implementational Issues of Support Vector Machines. Technical Report CSD-TR-96-18, Computational Intelligence Group, Royal Holloway, University of London, 1996.

[17] M. O. Stitson, J. A. E. Weston, A. Gammerman, V. Vovk and V. Vapnik. Theory of Support Vector Machines. Technical Report CSD-TR-96-17, Computational Intelligence Group, Royal Holloway, University of London, 1996.

[18] V. Vapnik, S. Golowich and A. Smola. 1997. Support Vector Method for Function Approximation, Regression Estimation, and Signal Processing. In: M. Mozer, M. Jordan, and T. Petsche (eds.): *Neural Information Processing Systems*, Vol. 9. MIT Press, Cambridge, MA, 1997.

[19] V. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York.

[20] G. Wahba. *Spline Models for Observational Data*. Series in Applied Mathematics, Vol. 59, SIAM, Philadelphia, 1990.

Appendix - Implementation Issues

[16] considers chunking.

[14] considers decomposition algorithm with guaranteed convergence to the global minimum.

Numerical Considerations

Hessian badly conditioned

zero order regularisation

sensitivity of solution to zero order regularisation.

The support vector algorithms were implemented in MATLAB.

Support Vector Classification

The optimisation problem can be expressed in matrix notation as,

$$\min_x \frac{1}{2}\alpha^T H\alpha + c^T\alpha \tag{60}$$

where

$$H = ZZ^T, \quad c^T = (-1,\ldots,-1) \tag{61}$$

with constraints

$$\alpha^T Y = 0, \quad \alpha_i \geq 0, i = 1,\ldots,l. \tag{62}$$

where

$$Z = \begin{bmatrix} y_1\mathbf{x}_1 \\ \vdots \\ y_l\mathbf{x}_l \end{bmatrix}, \quad Y = \begin{bmatrix} y_1 \\ \vdots \\ y_l \end{bmatrix} \tag{63}$$

The MATLAB implementation is given below:

```
function [nsv, alpha, b0] = svc(X,Y,ker,C)
%SVC Support Vector Classification
%
%  Usage: [nsv alpha bias] = svc(X,Y,ker,C)
%
%  Parameters: X      - Training inputs
%              Y      - Training targets
%              ker    - kernel function
%              C      - upper bound (non-separable case)
%              nsv    - number of support vectors
%              alpha  - Lagrange Multipliers
%              b0     - bias term
%
%  Author: Steve Gunn (srg@ecs.soton.ac.uk)
```

```
if (nargin <2 | nargin>4) % check correct number of arguments
  help svc
else

  n = size(X,1);
  if (nargin<4) C=Inf;, end
  if (nargin<3) ker='linear';, end
  epsilon = 1e-10;

  % Construct the H matrix and c vector

  H = zeros(n,n);
  for i=1:n
     for j=1:n
        H(i,j) = Y(i)*Y(j)*svkernel(ker,X(i,:),X(j,:));
     end
  end
  c = -ones(n,1);

  % Add small amount of zero order regularisation to
  % avoid problems when Hessian is badly conditioned.

  if (abs(cond(H)) > 1e+10)
    fprintf('Hessian badly conditioned, regularising ....\n');
    fprintf('    Old condition number: %4.2g\n',cond(H));
    H = H+0.00000001*eye(size(H));
    fprintf('    New condition number: %4.2g\n',cond(H));
  end

  % Set up the parameters for the Optimisation problem

  vlb = zeros(n,1);      % Set the bounds: alphas >= 0
  vub = C*ones(n,1);     %                 alphas <= C
  x0 = [ ];              % The starting point is [0 0 0   0]
  neqcstr = nobias(ker); % Set the number of equality constraints (1 or 0)
  if neqcstr
     A = Y';, b = 0;     % Set the constraint Ax = b
  else
     A = [];, b = [];
  end

  % Solve the Optimisation Problem

  st = cputime;

  if ( vlb == zeros(size(vlb)) & min(vub) == Inf & neqcstr == 0 )
     % Separable problem with Implicit Bias term
     % Use Non Negative Least Squares
     alpha = fnnls(H,-c);
  else
     % Otherwise
     % Use Quadratic Programming
     alpha = qp(H, c, A, b, vlb, vub, x0, neqcstr, -1);
  end

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


  % Compute the number of Support Vectors
  svi = find( abs(alpha) > epsilon);
  nsv = length(svi);

  if neqcstr == 0
     % Implicit bias, b0
     b0 = 0;
  else
     % Explicit bias, b0;
     % find b0 from pair of support vectors, one from each class
     classAsvi = find( abs(alpha) > epsilon & Y == 1);
     classBsvi = find( abs(alpha) > epsilon & Y == -1);
     nAsv = length( classAsvi );
     nBsv = length( classBsvi );
     if ( nAsv > 0 & nBsv > 0 )
        svpair = [classAsvi(1) classBsvi(1)];
        b0 = -(1/2)*sum(Y(svpair)'*H(svpair,svi)*alpha(svi));
     else
        b0 = 0;
     end
```

```
    end
  end
```

Support Vector Regression

The optimisation problem can be expressed in matrix notation as,

$$\min_x \frac{1}{2} x^T H x + c^T x \tag{64}$$

where

$$H = \begin{bmatrix} XX^T & -XX^T \\ \cdots & \cdots \\ -XX^T & XX^T \end{bmatrix}, \quad c = \begin{bmatrix} \varepsilon - Y \\ \cdots \\ \varepsilon + Y \end{bmatrix}, \quad x = \begin{bmatrix} \alpha^* \\ \cdots \\ \alpha \end{bmatrix} \tag{65}$$

with constraints

$$x \cdot (1,\ldots,1,-1,\ldots,-1) = 0, \quad \alpha_i, \alpha_i^* \geq 0, i = 1,\ldots,l. \tag{66}$$

where

$$X = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_l \end{bmatrix}, \quad Y = \begin{bmatrix} y_1 \\ \vdots \\ y_l \end{bmatrix} \tag{67}$$

The MATLAB implementation is given below:

```
function [nsv, beta, b0] = svr(X,Y,ker,e,C)
%SVR Support Vector Regression
%
%  Usage: alpha = svr(X,Y,ker,e,C)
%
%  Parameters: X      - Training inputs
%              Y      - Training targets
%              ker    - kernel function
%              e      - insensitivity
%              C      - upper bound (non-separable case)
%              nsv    - number of support vectors
%              beta   - Difference of Lagrange Multipliers
%              b0     - bias term
%
%  Author: Steve Gunn (srg@ecs.soton.ac.uk)

  if (nargin <3 | nargin>5) % check correct number of arguments
    help svr
  else

    n = size(X,1);
    if (nargin<5) C=Inf;, end
    if (nargin<4) e=0.05;, end
    if (nargin<3) ker='linear';, end
    epsilon = 1e-10; % tolerance for Support Vector Detection

    % Construct the H matrix and c vector

    H = zeros(n,n);
    for i=1:n
       for j=1:n
          H(i,j) = svkernel(ker,X(i,:),X(j,:));
       end
    end
    Hb = [H -H; -H H];
    c = [(e*ones(n,1) - Y); (e*ones(n,1) + Y)];

    % Add small amount of zero order regularisation to
```

```
    % avoid problems when Hessian is badly conditioned.
    % Rank is always less than or equal to n.
    % Note that adding to much reg will peturb solution

    if (abs(cond(Hb)) > 1e+10)
      fprintf('Hessian badly conditioned, regularising ....\n');
      fprintf('    Old condition number: %4.2g\n',cond(Hb));
      Hb = Hb+0.000000000001*eye(size(Hb));
      fprintf('    New condition number: %4.2g\n',cond(Hb));
    end

    % Set up the parameters for the Optimisation problem

    vlb = zeros(2*n,1);    % Set the bounds: alphas >= 0
    vub = C*ones(2*n,1);   %                 alphas <= C
    x0 = [ ];              % The starting point is [0 0 0   0]
    neqcstr = nobias(ker); % Set the number of equality constraints (1 or 0)
    if neqcstr
       A = [ones(1,n) -ones(1,n)];, b = 0;     % Set the constraint Ax = b
    else
       A = [];, b = [];
    end

    % Solve the Optimisation Problem

    st = cputime;

    if ( vlb == zeros(size(vlb)) & min(vub) == Inf & neqcstr == 0 )
       % Separable problem with Implicit Bias term
       % Use Non Negative Least Squares
       alpha = fnnls(Hb,-c);
    else
       % Otherwise
       % Use Quadratic Programming
       alpha = qp(Hb, c, A, b, vlb, vub, x0, neqcstr, -1);
    end

    fprintf('Execution time: %4.1f seconds\n',cputime - st);
    fprintf('|w0|^2    : %f\n',alpha'*Hb*alpha);
    fprintf('Sum alpha : %f\n',sum(alpha));

    % Compute the number of Support Vectors
    beta = alpha(n+1:2*n) - alpha(1:n);
    svi = find( abs(beta) > epsilon );
    nsv = length( svi );

    if neqcstr == 0
       % Implicit bias, b0
       b0 = 0;
    else
       % Explicit bias, b0;
       % compute using robust method of Smola
       % find b0 from average of support vectors with interpolation error e
       svbi = find( abs(beta) > epsilon & abs(beta) < C );
       nsvb = length(svbi);
       if nsvb > 0
          b0 = (1/nsvb)*sum(Y(svbi) + e*sign(beta(svbi)) + H(svbi,svi)*beta(svi));
       else
          b0 = (max(Y)+min(Y))/2;
       end
    end

  end
```

Toolbox

The toolbox can be downloaded from http://www.isis.ecs.soton.ac.uk/research/svm/svm.html