A Brief Note on Support Vector Machines

Saurabh Singal
Sankhyaa Research, Singapore

Acknowledgements

The author wishes to thank Prof. M. K. Singal, New Delhi and Dr M. Palaniswamy, Melbourne for their help and encouragement.

A friend has heard of a hugely successful fund called the Deep Haven Fund but wants us to check if it is an Event Driven fund or a Relative Value/ Non Directional funds. He will not invest in event driven strategies – he will only put his money in relative value strategies. Suppose we have run a regression on a set of hedge fund returns. Further suppose that we have identified some of these funds as Event Driven hedge funds, and some of the others as Relative Value hedge funds. How do we help our friend?

SVM classification will be a good tool to use. We start MATLAB, and using the swSVM software described in Appendix 1, we proceed as follows.

```
% Load the regression coefficients for the hedge funds. The last column is 1 for Event Driven and –1
%for Non-Directional % Relative Value Funds. No other types of funds are included in the file inputs.txt

funds=load('funds.txt');
deephaven=load('deephaven.txt');
kerneltype='rbf';sigma=1;
dh=swSVM(funds(:,1:end-1),funds(:,end),kerneltype ,deephaven,sigma)
```

The MATLAB output is

```
dh =      -1.00
```

We will see later how to interpret this output, which indicates that Deep Haven is a Non Directional Relative Value fund. We suggest that our friend put his money in the Deep Haven Fund.

Overview and Notation

The Support Vector Machine (SVM) is a technique for classification and regression. Originally the SVM was devised for binary classification, or classifying data into two types. Generalizations when there are more than two classes are relatively straightforward.

In order to familiarise the reader with the problem we are trying to solve and the terminology used, we will answer the following questions: What is Binary Classification? What is Linear Separation? What is Non-Linear Separation? What is a Linear Discriminant? We will show how the Support Vector Machine finds an optimal decision boundary in the case of linearly separable data. We will then discuss how the Support Vector Machine maps the patterns into a higher dimensional feature space such that linear separation in the higher dimensional feature space allows for non-linear separation in the original space. The use of kernel functions that allow this transformation will be also illustrated. The last section discusses the applications, which SVMs are being put to.

There are two Appendices. Appendix 1 describes a set of MATLAB programs that implements the SVM. Appendix 2 is the formulation of the Quadratic Programming Problem, which yields the Support Vector Machine solution.

Let us introduce the notation used in this note.

$\mathbf{R}^n$ is the real n-dimensional vector space. We will use **u, v, w,** and **x** to denote points in $\mathbf{R}^n$. These points are also called vectors or patterns in Machine Learning literature.

Each point belongs to one of two categories; a category is also called a class or type.

Each point $\mathbf{x}_i$ has a label y_i to denote which class $\mathbf{x}_i$ belongs to; $y_i = +1$ if $\mathbf{x}_i$ belongs to class 1 and $y_i = -1$ if $\mathbf{x}_i$ belongs to class 2. The choice of $+1$ and -1 for use as labels is both notationally convenient and also simplifies the calculations.

- ***What is Binary Classification?***
  Binary classification, as the name suggests, means classifying data into ***two*** categories. We are presented with some data points, or *training patterns*. We know for each of them, whether the pattern belongs to the first category or the second. Next, we are presented with some more data points but we do not know their respective classes. These new data points are called *test patterns*. The task at hand is to determine the category to which each test pattern belongs. Formally, given a set $\mathbf{x}_i$'s we wish to determine the corresponding $\mathbf{y}_i$'s. We only have the knowledge of the training patterns and their associated membership into either category. This process is called Binary Classification.

- ***What is Linear Separation? What is Non-Linear Separation? What is the Linear Discriminant?***

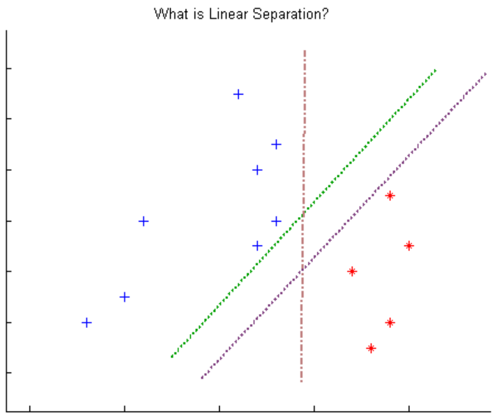


Figure 1. Red Asterisk markers and Blue Plus markers are patterns belonging to Class 1 and 2 respectively. Each of the three Straight Lines can separate the test patterns. This is an example of Linear Separation.

Look at Figure 1. The blue plus markers and the red asterisk markers represent training patterns belonging to class 1 and class 2 respectively. We arbitrarily label class 1 as "+1" and class 2 as "-1". We can draw a straight line that ***separates*** the patterns such that all patterns on one side of the line belong to class 1 and all patterns belonging to class 2 are on the other side of the line. In fact we can draw infinitely many such lines; we have shown three that do the job. This is an example of linearly separable data. If we draw a line that separates the two classes and are now asked to classify a new pattern, we will examine on which side of the line the pattern falls and classify it accordingly.

The notion of linear separation can be generalised to higher dimensions. In three dimensions, data that can be separated by a plane are linearly separable, for example.

Obviously, not all data are linearly separable. Figures 4 and 6 shown in later pages of this note are examples of points that are not linearly separable. Figure 6 shows a solid magenta curve that separates the patterns. Separation of points by a curve, which is not a straight line, is called non-linear separation.

The Linear Discriminant

A discriminant function is a function that lets us discriminate between different patterns. If **u** and **v** are two patterns and *g()* is a discriminant function, then knowledge of $g(\mathbf{u})$ and $g(\mathbf{v})$ will help us determine whether **u** and **v** are in the same class or not. If the said function is linear in the components of x, then it is called a linear discriminant. More formally, consider the function *g(),* $g(\mathbf{x})=\mathbf{w}^T\mathbf{x} + b=0$.

Here, the vector **w** is called the weight vector and it has the same dimensions as **x**.
Knowing the weight vector **w** and the constant parameter *b*, (called *bias term*), any pattern $\mathbf{x} = (x1, x2 \ldots xn)$ can be classified as belonging to class 1 or class 2 according to the following rule:

If $\mathbf{w}^T\mathbf{x} + b>=0$, classify **x** as belonging to class 1
If $\mathbf{w}^T\mathbf{x} + b<0$, classify **x** as belonging to class 2

In the case of two-dimensional **x** and **w,** $\mathbf{w}^T\mathbf{x} + b=0$ defines a straight line. Points on one side of this straight line will be classified as belonging to class 1; points on the other side of this line will be classified as belonging to class 2.

But there are an infinite number of straight lines that can linearly separate the data points; we can vary b to get parallel lines that will do the job. So we need to determine the "best" or optimal **w** and *b.*

To choose "good" **w** and *b*, we measure the distance $r(\mathbf{x})$ of **x** from the decision surface $g(\mathbf{x})=0$. The distance *r*, of a point **x** from the plane **P** specified by (**w**, *b*) is

$$r(\mathbf{x};\mathbf{w},b) = |g(\mathbf{x})|/||\mathbf{w}|| = |\mathbf{w}^T\mathbf{x} + b|/||\mathbf{w}||$$

When we talk of the distance from a point to a plane we mean the distance from **x** to the nearest point $\mathbf{x}_p$ that lies on the plane P.

Let us first introduce the term *margin of separation,* which we will denote by *M.* The margin of separation measures the distance between the two classes. It is shown in Appendix 2 that $M=2/||\mathbf{w}||$

The ***optimal separating hyperplane*** separates the two classes and maximizes the distance to the closest point from either class. This provides a unique solution to the separating hyperplane problem. By maximizing the margin between the classes, it leads to better classification.

Classification in the Linearly Separable Case.

Let us consider the problem of binary classification when the data are linearly separable. We are going to end up with a model which is easy to implement and very fast to compute. It will be straightforward to extend the model to case when the data are not linearly separable. Starting with the simpler, linearly separable case allows us to gain an understanding of the model quickly.

Suppose we have k training patterns, $(\mathbf{x_1}, \mathbf{x_2}, \dots, \mathbf{x_k})$ and their labels $(y_1, y_2, \dots, y_k)$.

$$\mathbf{x} \in \mathbf{R}^n, y \in \{+1, -1\}$$

Let us take n=2 for ease of visualization. We have shown two figures (Figure 1 and Figure 2), which show a scatter plot of the same data. The red star markers are patterns belonging to class 1 (or y = 1). The black plus markers are patterns belonging to class 2 (or y = -1). Each of the figures shows a solid line that acts as the separating hyperplane.

Obviously, there are many possible hyperplanes (straight lines) that can separate the data. Which is the one we should use? We want to choose a hyperplane that generalises well so that when the future patterns need to be classified, we do a good job. The key idea behind Support Vector Machines is that out of all the hyperplanes that can do the job (i.e., minimise training error) we should choose the one that has the maximal margin.

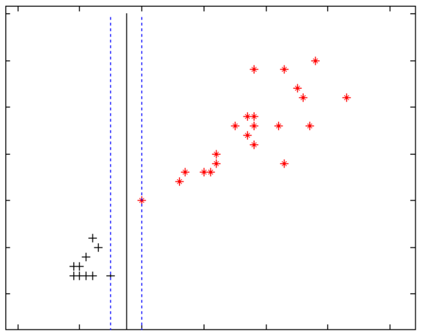


Figure 3: Separating Hyperplane (solid line) with Narrower Margin. Margin is the distance between the dotted lines

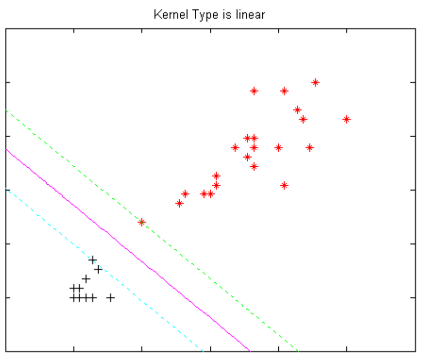


Figure 3: Separating Hyperplane (solid magenta line) with Wider Margin. Margin is the distance between the dotted lines

An SVM constructs a hyperplane $g(\mathbf{x}) = 0$ that will act as a decision surface in such a way that the margin of separation between the two classes is maximised. When we look at the two diagrams, we can see that the second diagram shows a hyperplane with a larger margin. Larger margin promises better performance on unseen data, in other words a larger margin leads to better generalisation. For details see Vapnik [2] which is the definitive work on Statistical Learning Theory. In our case, we want to find the parameters w $=[w_1\ w_2\ \ldots\ w_n\ ]^T$ and b of the decision function $d(\mathbf{x}, \mathbf{w}, \mathbf{b})$ given as

$$d(\mathbf{x}, \mathbf{w}, \mathbf{b}) = \mathbf{w}^T\mathbf{x} + b = \sum_{i=1}^{n} w_i x_i + b$$

After training is successful, using the weight vector **w** and the bias b, a pattern **x** is classified as by examining the sign of $d(\mathbf{x}, \mathbf{w}, \mathbf{b})$.

If $d(\mathbf{x_p}, \mathbf{w}, b) >= 0$, pattern $\mathbf{x_p}$ belongs to class 1, ($\mathbf{y_p}=1$)
If $d(\mathbf{x_p}, \mathbf{w}, b) < 0$, pattern $\mathbf{x_p}$ belongs to class 2, ($\mathbf{y_p}= -1$)

The geometric interpretation is that the equation $\mathbf{w}^T\mathbf{x} + b = 0$ divides the input space into two half spaces. As our input is two dimensional, $\mathbf{w}^T\mathbf{x} + b = 0$ is the equation of a straight line and it will divide $\mathbf{R}^2$ into two half spaces, the elements of class 1 lie in one half space and the elements of class 2 lie in the other half space. This notion is easily extended to dimensions higher than two.

The decision boundary is a hyperplane; more precisely it is a separating hyperplane in $\mathbf{R}^2$. Before we describe how the weight vector **w** and bias term b can be estimated, we will introduce the concept of a canonical hyperplane. First we note that if $d(\mathbf{x},\mathbf{w},b)$ is separating function, then for any k>0, $d(\mathbf{x}, k\mathbf{w}, kb)$ is also a correct decision function.
A hyperplane is canonical with respect to the data **x** if

$$\min_{\mathbf{x}_i \in X} \left| \mathbf{w}^T \mathbf{x}_i + b \right| = 1$$

As the parameters (**w**,b) and (k**w**, kb) describe the same hyperplane, it is important to talk in terms of canonical hyperplanes. The ***optimal canonical hyperplane*** is the canonical hyperplane with the maximal margin. The optimal canonical hyperplane will yield the decision boundary that the Support Vector Machine seeks.

The solid magenta line in Figure 3 is the decision boundary that the SVM obtains. The dotted green and cyan lines are the margin. Points on the margin are called ***Support Vectors***. The MATLAB implementation of SVM described in Appendix 1 was used for this example.

Classification when Data are Linearly Non-Separable.

The true power of SVMs comes into play when we have data points that are not separable by a linear decision surface. A canonical example is the XOR function (exclusive OR), which takes a value of 0 when both its inputs are the same and takes a value of 1 otherwise. In our notation this becomes

x=(x1,x2)	*y*
(1,1)	-1
(-1,-1)	-1
(1,-1)	1
(-1,1)	1

Table 1: The XOR Gate

We show this in Figure 3, where the red asterisk symbols marks the two points for which y_i=-1 and the blue plus symbol marks points for which y_i= 1.

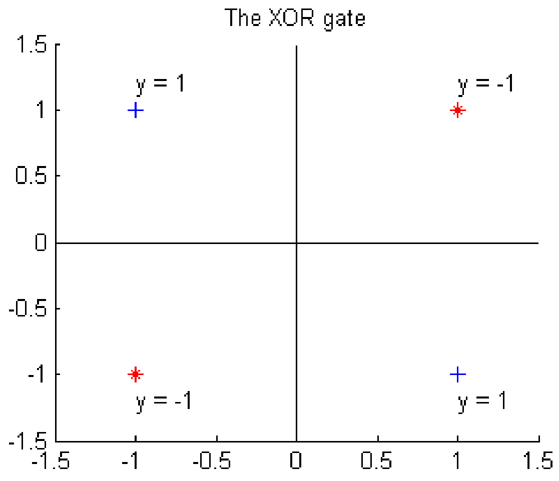


Figure 4: The XOR Gate. There is no straight line that can separate the two classes.

These points cannot be linearly separated, that is, we cannot draw any straight line that will separate the classes. But there is a trick we can use; we define *x3= x1.x2* and then augment the 2 dimensional input vectors by this third number, *x3*. In our XOR example, we get

x=(x1,x2,x1.x2)	*y*
(1,1,1)	-1
(-1,-1,1)	-1
(1,-1,-1)	1
(-1,1,-1)	1

Table 2: The modified XOR Gate

Now it is easy to see that these points are linearly separable in $\mathbf{R}^3$; the plane z=0 is one of infinitely many linear decision surfaces that can separate the data points. This is depicted in Figure 4. The blue dots are the points (1, -1,1) and (-1,1,1), corresponding to the points in the two dimensional space for which the y_i is 1. The red dots are the points (1,1, -1) and (-1, -1, -1), corresponding to the points in the two dimensional space for which the y_i is –1.The patterns were separated by examining the sign of *uv* in our example, which is a non-linear function of *u* and *v*, but is a linear function of *uv*. Thus a linear separator in the feature space could act as a non-linear separator in the original space or the data space.

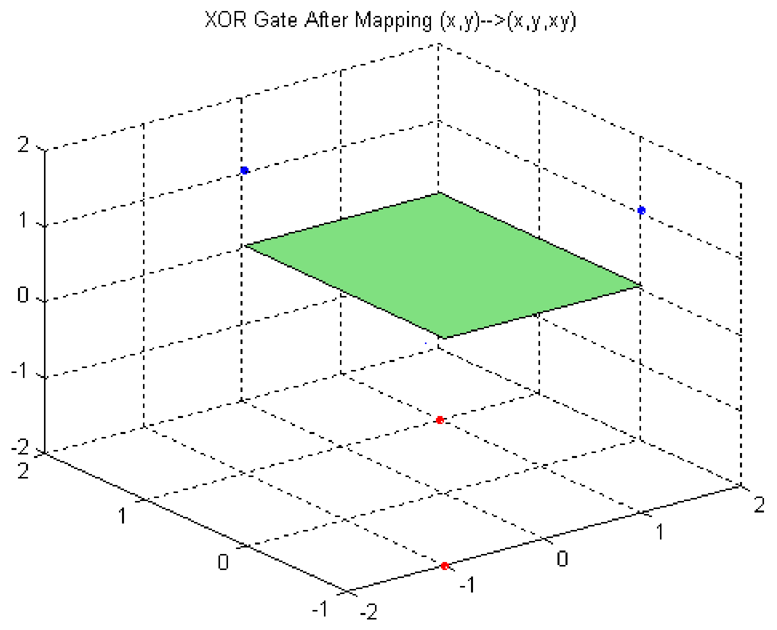


Figure 5: Separation of the XOR Gate After Mapping to Higher Dimension.

This brings us to another important idea of SVMs- the use a non-linear function to map the training vectors or data points into a higher dimensional space. This higher dimensional space is called the *Feature Space*. We find and use a linear decision surface in the feature space, and this allows for non-linear separation in the original space.

In our example, we used the mapping $\boldsymbol{\Psi}(u,v) \mapsto (u,v,uv)$ to map the two dimensional pattern space to a three dimensional feature space.

The strategy that allows us to devise non-linear classifiers is to map input vectors $\mathbf{x} \in \mathbf{R}^n$ into vectors $\mathbf{z}$ of a higher dimensional feature space F. The function that performs this mapping will be denoted by $\mathbf{\Phi}$. Formally,
$\mathbf{z} = \mathbf{\Phi}(\mathbf{x})$, where $\mathbf{\Phi}$ is a mapping $\mathbb{R}^n \to \mathbb{R}^f; f > n$

Kernel Functions

We have shown in Appendix 2 that to find the optimal linear decision boundary, we need to minimize

$$L_d(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j$$

To find a non-linear decision boundary, we have to replace
$\mathbf{x}_i^T \mathbf{x}_j$ by $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{\Phi}^{\mathrm{T}}(\mathbf{x}_i)\mathbf{\Phi}(\mathbf{x}_j)$ in the above expression.

The function *K( )* is called a *kernel function*. The kernel function is allowing us to compute an inner product in the higher dimensional feature space. Moreover, if we can compute the inner product, $K(\mathbf{u},\mathbf{v})$ we do not need to know or compute $\mathbf{\Phi}(\mathbf{u})$ and $\mathbf{\Phi}(\mathbf{v})$ themselves; we just need to be able to compute the inner product.

For a function *K* to have the right form as a kernel, it must satisfy a set of conditions called the Mercer conditions, which are:

1. $K(\mathbf{u},\mathbf{v}) = \mathbf{\Phi}^{\mathrm{T}}(\mathbf{u})\mathbf{\Phi}(\mathbf{v})$

2. $K(\mathbf{u},\mathbf{v}) = K(\mathbf{v},\mathbf{u})$

3. $\int_a^b \int_a^b K(\mathbf{u},\mathbf{v}) \psi(\mathbf{u}) \psi(\mathbf{v}) \, d\mathbf{u} \, d\mathbf{v} \geq 0$, for all functions ψ for which ψ^2 is integrable over $[\mathbf{a},\mathbf{b}]$

Some commonly used kernel functions are listed in Table 3.

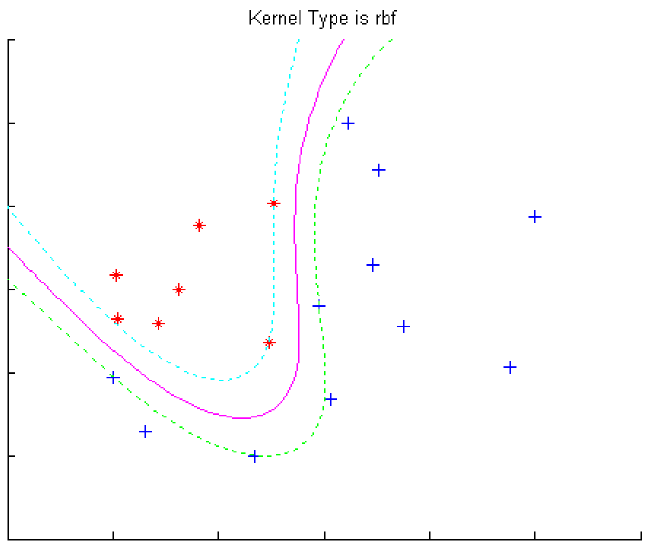
Name of Kernel Function	Definition
Linear	$K(\mathbf{u},\mathbf{v}) = \mathbf{u}^T \mathbf{v}$
Polynomial of degree *d*	$K(\mathbf{u},\mathbf{v}) = (\mathbf{u}^T \mathbf{v} + 1)^d$
Gaussian Radial Basis Function (RBF)	$K(\mathbf{u},\mathbf{v}) = e^{-\frac{1}{2}[(\mathbf{u}-\mathbf{v})^T \Sigma^{-1} (\mathbf{u}-\mathbf{v})]}$

Sigmoid	$K(\mathbf{u}, \mathbf{v}) = \tanh[\mathbf{u}^T \mathbf{v} + b]$

Table 3. Some Commonly Used Kernel Functions

The SVM produces a non-linear boundary in the original pattern space by constructing a linear boundary in a large, transformed version of the feature space.

Figure 6 presents an example of non-linear separation using the SVM. The MATLAB programs described in Appendix 1 are used for this the computations and plotting. The red asterisk markers in Figure 6 are patterns belonging to class 1 and the blue plus symbols are from class 2. Using Gaussian RBF kernel, the SVM have achieved non-linear separation. The magenta solid line is the non-linear decision boundary, and the margin is shown by the dotted cyan and the dotted green curves. The points on the margin are the support vectors.



.

Figure 6: Mapping to Higher Dimensional Feature Space Using RBFs Permits Non-Linear Separation.

Applications of Support Vector Machines

Support Vector Machines are finding many uses in pattern recognition and classification tasks. We list some of them here, but for a more detailed discussion, see Christianini [2].

The task of text categorisation refers to the classification of natural text documents into a fixed number of predefined categories based on their content. This is important in email filtering, web searching, sorting documents by topic and classification of news stories.

The automatic categorisation of images is gaining importance in medical applications. Hand-written digit recognition was one of the first real world task on which Support Vector Machines were tested. The problem on which the test was carried is used as a benchmark for classification schemes. Support Vector Machines did as well on the problem as other classification algorithms that were designed specifically for this problem. It is remarkable that SVM performed as well as these other systems without any detailed prior knowledge.

Bioinformatics is another area where SVMs are being applied in many different ways.
One important area of research in bioinformatics is to predict the features of a protein based on its amino acid sequence. One approach involves relating new protein sequences to proteins whose properties are already known. Similarity between proteins is called Protein Homology. An SVM based protein homology detection system handily outperformed state-of-the art protein homology detection systems.

SVMs are being used in the automatic categorisation of gene expression data from DNA microarrays. As the volume of genomics data grows, it is becoming important to have automated means of assigning functions to genes. SVMs are being applied very successfully in gene expression data for classifying unseen genes and have outperformed various systems that use Fisher's Linear Discriminant or decision trees and other less recent techniques.

Bibliography

[1] Christianini, Nello and John Shawe-Taylor (2000), An *Introduction to Support Vector Machines,* Cambridge University Press, Cambridge.

[2] Vapnik, Vladimir N (1999), *The Nature of Statistical Learning*, Springer-Verlag, New York.

Appendix 1: The MATLAB *swSVM* Software.

We have a programmed a simple SVM implementation in MATLAB. The example and the figures in the preceding pages where obtained using this software. There are only two choices for the kernel function that have been implemented, namely, Gaussian RBF and Linear.

Description of the Functions.

There are five MATLAB functions, and by their function we have broken them into three categories.

Functions for SVM Training and Classification.

There are three functions that implement the SVM- ***swSVM, swquad*** and ***swSVMclassify***.

The function that solves the QP for obtaining the SVM solution is ***swquad***.

The function ***swSVMclassify*** classifies test patterns, using the solution obtained previously by *swquad*.

The function ***swSVM*** is like a main function that accepts as inputs the training patterns and their labels; the test patterns, the choice of the kernel and the sigma parameter (only needed for the RBF kernel). It calls *swquad* for solving the Quadratic Program to perform SVM separation and then calls *swSVMclassify,* which classifies the test patterns. If the data are two dimensional, swSVM also plots the training patterns, the margin and the decision boundary by calling ***swPlot***.

Plotting Related Functions

The function ***swscatter*** does a scatter plot of the training patterns; and it outputs the patterns of category 1 as blue plus markers and the patterns of category 2 as red asterisk. (We assign y=1 to the patterns of category 1 and y= -1 to the patterns of category 2, but this can be reversed with no loss of generalization). The function ***swPlot*** is used to plot the training patterns and the decision boundary as well as the margin of the SVM. This function uses the outputs of the swquad function. The function swPlot calls the function ***swscatter***.

Helper functions

There is one helper function ***swscale***, which is called by other functions for scaling the training patterns before plotting the margin and decision boundaries.

A Sample Session of the swSVM MATLAB software

We will show an example of non-linear separation using the Gaussian RBF as the kernel. The data is the same used as in Figure 6. After obtaining the decision boundary, we will pick two arbitrary patterns as test patterns and classify them using the SVM e have obtained.

In the lines that follow, we will use the `Courier` font to denote MATLAB commands and the *Tohama* font for comments. The output from MATLAB is in the **Arial** font

```
% Load the data from the file nonlin.txt
% The first two columns are the patterns and the third column contains the associated   labels for each
% pattern. The label is 1 for patterns of class 1 and –1 for patterns of class 2.

data=load('nonlin.txt');

%Put the Patterns into variable x and
x=data(:,1:2);

% Put the class labels into the variable y
y=data(:,3);

% Put choose RBF as the choice of kernel since we will perform non-linear separation
kerneltype='rbf';

%Use Sigma =1
sigma=1;

% Let us perform a non-linear separation of the data using swquad
[W0,b0, alpha]=swquad(x,y,kerneltype,sigma );

Optimization terminated successfully.
The number of support vectors is 5
The support Vectors are points number:
 3
 7
 9
16
17

% Now we will plot the decision boundary, the margin and the training patterns.
% We use blue plus markers for class 1 patterns (y= 1) and red asterisk   markers for class 2 patterns
% (y= -1)

swPlot(x,y,'rbf',alpha, W0,b0 ,sigma);

% Now let us pick two arbitrary points and try to classify them.
hold on
```

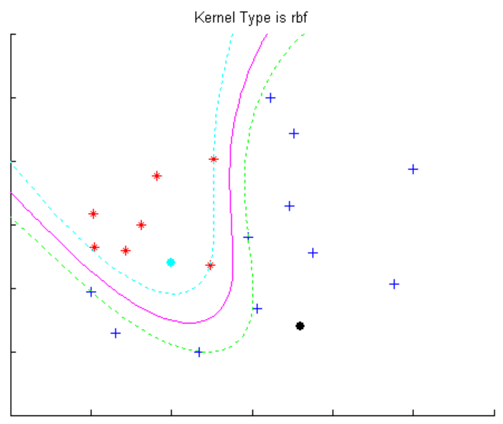


Figure 7. Non-linear Separation using Gaussian RBF Kernel. Blue Plus markers are class 1 patterns. Red Asterisk markers are class 2 patterns. The two test patterns are shown as cyan and black filled circles.

```
%Choose a point (pattern1) and Plot pattern 1 in Cyan colour on the existing plot
pattern1= [-0.5,  -0.3];
scatter(pattern1(1),pattern1(2), 401,'c.')

%Choose a point (pattern1) and Plot pattern 1 in Black colour on the existing plot
pattern2= [0.3,  -0.8];
scatter(pattern2(1),pattern2(2), 401,'k.')

%Now store pattern1 and pattern2 as the first and second rows of the matrix test
test=[pattern1;pattern2];

% Classify test patterns
class=swSVMclassify(alpha, b0, x,y,test,kerneltype,sigma)
```

```
class =
       -1.00
        1.00
```

We see that pattern 1 is classified as belonging to class 2 (i.e., label = -1) and pattern 2 as class 1 (label =1). From the positions of the cyan and black markers on the figure this is what one might expect. We could also have done the task of finding the SVM, plotting the training patterns and decision boundary; and classifying the test patterns by the use of the function *swSVM*. The results would be the same as calling *swquad*, *swPlot* and *swSVMclassify* in sequence as we have shown in the preceding lines.

```
class =swSVM(x,y,kerneltype ,test,sigma)
```

Appendix 2: Formulation of the Quadratic Programme for the SVM

We will derive the solution for the optimal canonical separating hyperplane when the data are linearly separable. We note that this hyperplane is a canonical separating hyperplane with the maximal margin.

The margin $M = (\mathbf{x}_1 - \mathbf{x}_2)_{\mathbf{w}}$.
Here the subscript w denotes the projection of the vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ onto the weights vector direction. Taking projections along **w**, we get
$D_1 = \|\mathbf{x}_1\| \cos(\alpha)$,
$D_2 = \|\mathbf{x}_2\| \cos(\beta)$,
$M = D_1 - D_2$.

where α and β are the angles between **w** and **x₁** and **w** and **x₂** respectively. We know that

$$\cos(\alpha) = \frac{\mathbf{x}_1^T \mathbf{w}}{\|\mathbf{x}_1\| \|\mathbf{w}\|}$$
$$\cos(\beta) = \frac{\mathbf{x}_2^T \mathbf{w}}{\|\mathbf{x}_2\| \|\mathbf{w}\|}$$

Substituting, this leads to $M = \dfrac{\mathbf{x}_1^T\mathbf{w} - \mathbf{x}_2^T\mathbf{w}}{\|\mathbf{w}\|}$, and as $\mathbf{x}_1$ and $\mathbf{x}_2$ are support vectors satisfying $y_j \left| \mathbf{w}^T \mathbf{x}_j + b \right| = 1,\ j = 1, 2$
we have $\mathbf{w}^T \mathbf{x}_1 + b = 1,\ \mathbf{w}^T \mathbf{x}_2 + b = -1$
and finally we get $M = \dfrac{2}{\|\mathbf{w}\|}$. (A.1)
We could also get this result using the fact that the distance D between a support vector x1 and a canonical separating line is equal to half the margin M and therefore $D = \dfrac{M}{2} = \dfrac{\left|\mathbf{w}^T \mathbf{x}_2 + b\right|}{\|\mathbf{w}\|} = \dfrac{1}{\|\mathbf{w}\|}$, from where $M = \dfrac{2}{\|\mathbf{w}\|}$, as before. Therefore to maximise the margin, *M*, we need to minimise $\|\mathbf{w}\| = \sqrt{w_1^2 + w_2^2 ... + w_n^2}$.

The optimal canonical separating hyperplane with the maximal margin will specify support vectors – that is the training points closest to the OCSH by
(A.2)
where N_{sv} denotes the total number of support vectors.
At the same time, all training points must satisfy the following inequalities. $y_j [\mathbf{w}^T \mathbf{x}_j + b] = 1,\ j = 1 \,..\, l$ (A.3)

Thus to find the optimal separating hyperplane with the maximal margin, we need to minimize $\|\mathbf{w}\|$, which is the same as minimising $\|\mathbf{w}\|^2$, subject to (A.3). You will recognize that this is a standard non-linear optimisation problem with inequality constraints, which can be solved by the method of Lagrange multipliers.

Let $L(\mathbf{w},b,\alpha)=\frac{1}{2}\mathbf{w}^{\mathbf{T}}\mathbf{w}-\sum_{i=1}^{l}\alpha_i\{y_j[\mathbf{w}^T\mathbf{x}_j+b]-1\}$ (A.4)

where the α_i's are the Lagrange multipliers. The Lagrangian L in (a.4) is to be minimized with respect to **w** and *b* and maximized with respect to the non-negative α_i's. Instead of solving the problem in the primal space (the space of w and b) it is more insightful to solve the problem in the dual space (the space of the α_i's). Applying the Karush-Kuhn-Tucker conditions, at the optimal solution ($\mathbf{w}_0$, b_0, α_0) the derivatives of the Lagrangian with respect to the primal variables will vanish, so that

$$\frac{\partial L}{\partial \mathbf{w}_0}=0, \text{ or } \mathbf{w}_0=\sum_{i=1}^{l}\alpha_i y_i \mathbf{x}_i \qquad \text{(A.5)}$$

$$\frac{\partial L}{\partial b_0}=0, \text{ or } \sum_{i=1}^{l}\alpha_i y_i=0 \qquad \text{(A.6)}$$

Applying the complementarity conditions, we have

$$\alpha_i\{y_j[\mathbf{w}^T\mathbf{x}_j+b]-1\}=0, \quad i=1..l \qquad \text{(A.7).}$$

Substituting (A.5) and (A.6) into (A.4) gives us

$$L_d(\alpha)=\sum_{i=1}^{l}\alpha_i-\frac{1}{2}\sum_{i=1}^{l}\alpha_i\alpha_j y_i y_j \mathbf{x}_i{}^{\mathbf{T}}\mathbf{x}_j \qquad \text{(A.8)}$$

We need to maximise the dual Lagrangian $L_d(\alpha)$ with respect to the non-negative α_i's

$$\alpha_i>0, \ i=1..l \qquad \text{(A.9)}$$

The dual Lagrangian $L_d(\alpha)$ is expressed only in terms of the training data, and it depends on the scalar product of the input training patterns - $\mathbf{x_i x_j}$. This is very important because we will see that instead of using $\mathbf{x_i x_j}$ we will be able to use other types of inner products.

Our formulation is a standard Quadratic Programming problem. We can put this in matrix notation.

Maximize $L_d(\alpha) = -0.5\boldsymbol{\alpha}^T \mathbf{H}\boldsymbol{\alpha} + \mathbf{f}^T\boldsymbol{\alpha}$ (QP.1)

subject to

$\mathbf{y}^T\boldsymbol{\alpha} = 0$, (QP.2)

$\boldsymbol{\alpha} \geq \mathbf{0}$ (QP.3)

where **H** denotes the Hessian matrix ($\mathbf{H}_{ij} = y_i y_j x_i x_j$, or $y_i y_j \mathbf{x}_i^T \mathbf{x}_j$) and **f=1** is a unit vector; **f**= [1 1 1… 1].

After we find out the solution $\boldsymbol{\alpha_0}$ of the QP, we can find the parameters $\mathbf{w}_0$ and *b*, as follows

$$\mathbf{w}_0 = \sum_{i=1}^{l} \alpha_{0i} y_i \mathbf{x}_i, \qquad i = 1..l \tag{A.10}$$

$$b0 = \frac{1}{N_{sv}}\left(\sum_{s=i}^{N_{sv}}\left(\frac{1}{y_s} - \mathbf{x}_s{}^T\mathbf{w}_0\right)\right), \qquad s = 1..N_{sv} \tag{A.11}$$

When usng a kernel function *K*, we will replace $\mathbf{x}_i^T\mathbf{x}_j$ by $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{\Phi}^{\mathrm{T}}(\mathbf{x}_i)\mathbf{\Phi}(\mathbf{x}_j)$ in A.8 and subsequently.