Support Vector Machines

School of Informatics, University of Edinburgh

Chris Williams

October 2008

Support Vector Machines (SVMs) are a relatively new concept in supervised learning, but since the publication of [4] in 1995 they have been applied to a wide variety of problems. In many ways the application of SVMs to almost any learning problem mirrors the enthusiasm (and fashionability) that was observed for neural networks in the second half of the 1980's.

The ingredients of the SVM had, in fact, been around for a decade or so, but they were not put together until the early 90's. The two key ideas of support vector machines are

(i) The maximum margin solution for a linear classifier.

(ii) The "kernel trick"; a method of expanding up from a linear classifier to a non-linear one in an efficient manner.

Below we discuss these key ideas in turn, and then go on to consider support vector regression and some example applications of SVMs. Further reading on the topic can be found in [3], [5], and [8].

1 Linear Classifiers

There are two cases that must be considered. In section 1.1 we consider the case when the data is linearly separable; in section 1.2 the extension to the non-separable case is made.

1.1 The Separable Case

Consider the arrangement of data points shown in Figure 1(a); it is clear that the $\times$'s and o's can be linearly separated. Let the decision boundary be defined by $\tilde{\mathbf{w}}.\mathbf{x} + w_0 = 0$, and let $\mathbf{w} = (\tilde{\mathbf{w}}, w_0)$. Running the perceptron learning algorithm on this data would return a weight vector $\mathbf{w}$ that would classify the training examples correctly. However, there is a whole version space of weight vectors that give rise to the same classification of the training points. The SVM algorithm chooses a particular weight vector, that which gives rise to the "maximum margin" of separation (as explained below).

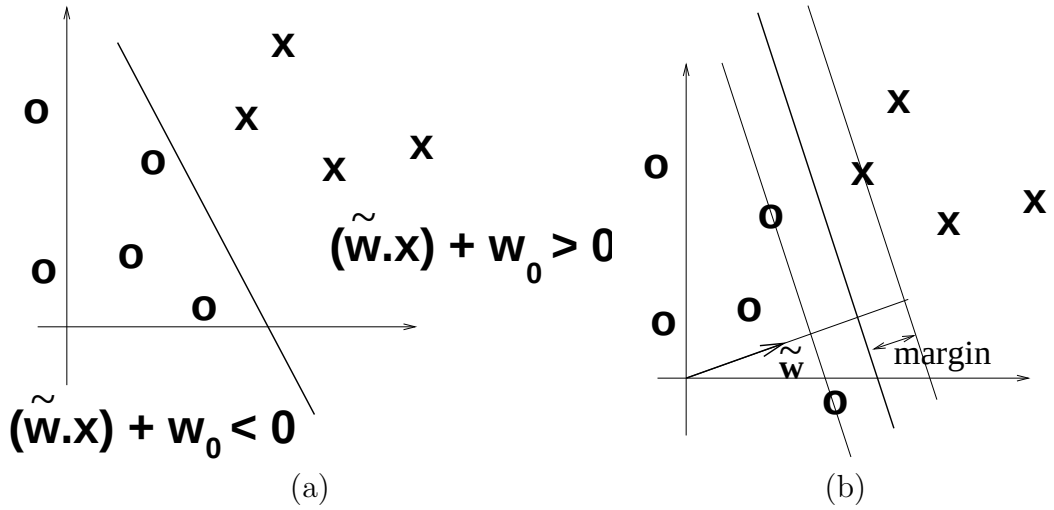

Figure 1: (a) A separating hyperplane. (b) The maximum margin separating hyperplane.

Let the training set be pairs of the form $(\mathbf{x}_i, y_i)$ for $i = 1, \ldots, n$, where $y_i = \pm 1$. Let the perpendicular distance from the hyperplane to the nearest $+1$ class point be denoted d_+, and similarly d_- for the -1 class. The margin is defined as $\min(d_+, d_-)$ and the SVM solution looks for the weight vector that maximizes this. Note that if the equation $f(\mathbf{x}) = \tilde{\mathbf{w}}.\mathbf{x} + w_0$ defines a discriminant function (so that the output is $\text{sgn}(f(\mathbf{x}))$), then the hyperplane $c\tilde{\mathbf{w}}.\mathbf{x} + cw_0$ defines the same discriminant function for any $c > 0$. Thus we have the freedom to choose the scaling of $\mathbf{w}$ so that $\min_{\mathbf{x}_i} |\tilde{\mathbf{w}}.\mathbf{x}_i + w_0| = 1$. Then we have

$$\mathbf{x}_i.\tilde{\mathbf{w}} + w_0 \geq +1 \qquad \text{for } y_i = +1, \tag{1}$$

$$\mathbf{x}_i.\tilde{\mathbf{w}} + w_0 \leq -1 \qquad \text{for } y_i = -1. \tag{2}$$

These can be combined into one set of inequalities

$$y_i(\mathbf{x}_i.\tilde{\mathbf{w}} + w_0) \geq 1 \qquad \text{for } i = 1, \ldots, n. \tag{3}$$

It is clear by considering the geometry that for the maximum margin solution $d_+ = d_-$, so that there is at least one data point in each class for which $y_i(\mathbf{x}_i.\tilde{\mathbf{w}} + w_0) = 1$. Consider a point $\mathbf{x}_+$ for which the equality in equation 1 holds; this gives $\mathbf{x}_+.\tilde{\mathbf{w}} + w_0 = 1$. Similarly for a point $\mathbf{x}_-$ for which equation 2 holds we obtain $\mathbf{x}_-.\tilde{\mathbf{w}} + w_0 = -1$. Let these two hyperplanes be denoted H_+ and H_- respectively. The perpendicular distance $(d_+ + d_-)$ between the two hyperplanes can be calculated as $\hat{\tilde{\mathbf{w}}}.(\mathbf{x}_+ - \mathbf{x}_-) = 2/||\tilde{\mathbf{w}}||$, where $\hat{\tilde{\mathbf{w}}}$ is the unit vector $\tilde{\mathbf{w}}/||\tilde{\mathbf{w}}||$. Thus to maximize the margin we can minimize $||\tilde{\mathbf{w}}||^2$ subject to the constraints in equation 3.

This constrained optimization problem can be set up using Lagrange multipliers, and solved using numerical methods for quadratic programming[1] problems (see [3] for details).

[1]A quadratic programming problem is an optimization problem where the objective function is quadratic and the constraints are linear in the unknowns.

The form of the solution is

$$\tilde{\mathbf{w}} = \sum_i \alpha_i y_i \mathbf{x}_i, \tag{4}$$

where the α_i's are non-negative coefficients determined numerically. Notice that the solution is a linear combination of the $\mathbf{x}_i$'s. (Recall that the perceptron learning rule started from $\mathbf{0}$ also gives a solution that is a linear combination of the $\mathbf{x}_i$'s.) The key feature of equation 4 is that α_i is zero for every $\mathbf{x}_i$ *except* those which lie on the hyperplanes H_+ or H_-; these points are called the *support vectors*. The fact that not all of the training points contribute to the final solution is referred to as the *sparsity* of the solution. The support vectors lie closest to the decision boundary. Note that if all of the other training points were removed (or moved around, but not crossing H_+ or H_-) the same maximum margin hyperplane would be found. The optimization problem for finding the α_i's is convex, i.e. there are no local minima. This is in contrast to the optimization problem for neural networks, where there are local minima.

To make predictions for a new input $\mathbf{x}$ we compute

$$g(\mathbf{x}) = \text{sgn}\left(\tilde{\mathbf{w}}.\mathbf{x} + w_0\right) \tag{5}$$

$$= \text{sgn}\left(\sum_{i=1}^{n} \alpha_i y_i (\mathbf{x}_i.\mathbf{x}) + w_0\right). \tag{6}$$

Notice that $\mathbf{x}$ enters this expression only on terms of the inner product $\mathbf{x}.\mathbf{x}_i$. This will be important when the kernel trick is discussed below.

1.2 The Non-Separable Case

In section 1.1 we discussed the separable case. We know that there are many cases in which the data is not linearly separable. In this case the perceptron learning algorithm simply does not terminate. However, we can set up an objective function that trades off misclassifications against minimizing $||\tilde{\mathbf{w}}||^2$ to find an optimal compromise. To do this we add a "slack" variable $\xi_i \geq 0$ for each training example and require that

$$\tilde{\mathbf{w}}.\mathbf{x}_i + w_0 \geq 1 - \xi_i \quad \text{for } y_i = +1 \tag{7}$$

$$\tilde{\mathbf{w}}.\mathbf{x}_i + w_0 \leq -1 + \xi_i \quad \text{for } y_i = -1. \tag{8}$$

(The terminology of *slack variables* comes from optimization theory.) The idea is that we do allow the constraints of equations 1 and 2 to be violated, but only if we pay a "price". In fact J, the function to be optimized, is given by

$$J = ||w||^2 + C(\sum_{i=1}^{n} \xi_i^k) \tag{9}$$

where C is the parameter that determines the relative contributions of the slack variables and $||\tilde{\mathbf{w}}||^2$. k is taken to be an integer, and is normally set to $k = 1$ in SVMs. The solution is again given by

$$\tilde{\mathbf{w}} = \sum_i \alpha_i y_i \mathbf{x}_i, \tag{10}$$

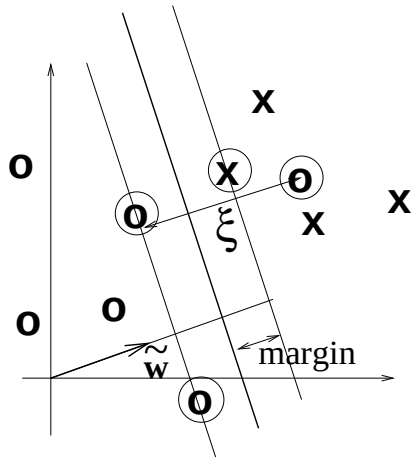

Figure 2: A non-separable example. The data points ringed are the support vectors.

where the α_i's are obtained from a (different) quadratic programming problem. In this case the support vectors (those with $\alpha_i \neq 0$) are not only those data points which lie on the separating hyperplanes, but also those that have non-zero ξ_i. This can occur in two ways:

- The data point falls in between H_+ and H_- but on the correct side of the decision surface,
- The data point falls on the wrong side of the decision surface.

A schematic of the non-separable case is shown in Figure 2.

2 Feature Space and the Kernel Trick

How can we generalize the methods of section 1 to give us a non-linear discriminant function? It turns out this can be achieved in a remarkably simple fashion using a trick from [1]. First notice that the only way that the data points appear in the training problem is on the form $\mathbf{x}_i.\mathbf{x}_j$, and the only way they appear in the testing phase with a new input $\mathbf{x}$ is as $\mathbf{x}_i.\mathbf{x}$. Suppose that we have mapped an input $\mathbf{x}$ into some other (possibly infinite dimensional) space $\mathcal{F}$ of dimensionality $N_{\mathcal{F}}$ with the mapping $\boldsymbol{\phi} : \mathbb{R}^d \rightarrow \mathcal{F}$. We call $\mathcal{F}$ the feature space defined by the mapping $\boldsymbol{\phi}$. The maximum margin algorithm can construct a separating hyperplane in the feature space; to do this it will need to evaluate inner products in feature space of the form $\boldsymbol{\phi}(\mathbf{x}_i).\boldsymbol{\phi}(\mathbf{x}_j)$. If there were a function (called a kernel function) so that $k(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\phi}(\mathbf{x}_i).\boldsymbol{\phi}(\mathbf{x}_j)$ then we only need to use k in the algorithm (for both training and testing phases).

One example of a feature space is the mapping (for a two-dimensional input, i.e. $d = 2$)

$$\boldsymbol{\phi}(\mathbf{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}. \tag{11}$$

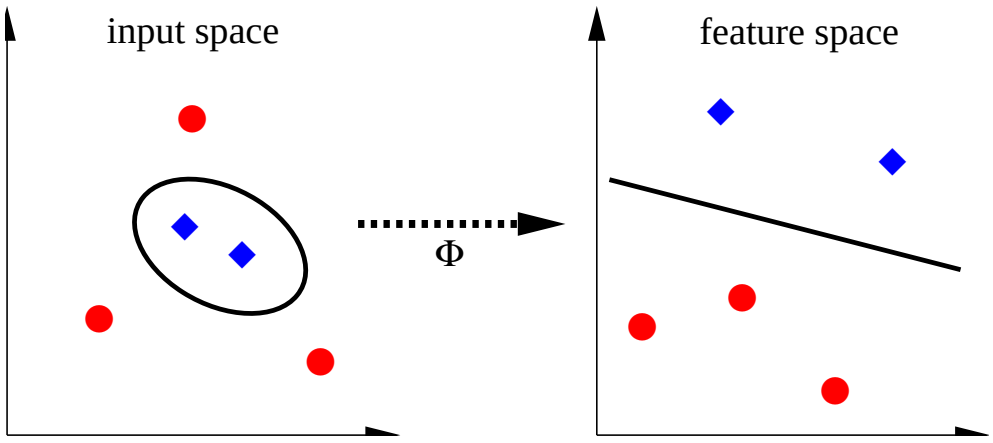

Figure 3: In the input space the circles and diamonds are not linearly separable; however, in feature space they are.

It can easily be verified that $\boldsymbol{\phi}(\mathbf{x}_i).\boldsymbol{\phi}(\mathbf{x}_j) = (\mathbf{x}_i.\mathbf{x}_j)^2$. Another example of a kernel is the $k(\mathbf{x}_i, \mathbf{x}_j) = \exp -||\mathbf{x}_i - \mathbf{x}_j||^2/2\sigma^2$. In fact this corresponds to an infinite-dimensional feature space, so an explicit expansion into $\mathcal{F}$ and computation there is impossible.

The objective of the transformation to feature space is illustrated in Figure 3. A problem that is not linearly separable in the original $\mathbf{x}$ space can sometimes be made so by a transformation to a higher-dimensional feature space.

In the testing phase, equation 6 is modified to

$$g(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^{n} \alpha_i y_i (\boldsymbol{\phi}(\mathbf{x}_i).\boldsymbol{\phi}(\mathbf{x})) + w_0\right) \tag{12}$$

$$= \text{sgn}\left(\sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + w_0\right), \tag{13}$$

with the α_i's and w_0 determined by a quadratic programming problem.

A possible concern about expanding the input vector into feature space is that we will need (in theory) to learn weights for all $N_{\mathcal{F}}$ dimensions. In general we expect that to learn more parameters more data will be needed. However the minimization of the $||\tilde{\mathbf{w}}||^2$ term corresponds to a regularization of these parameters and it turns out that the expansion into feature space can be done both safely and computationally efficiently.

2.1 Choosing the kernel and C

We have so far concentrated on solving the optimization problem for SVMs, given a particular kernel function k and choice of C. Of course in practice it will usually not be obvious what choices should be made for these. There are theoretical VC dimension bounds for SVMs that relate properties of the probability distribution for the generalization performance to a given training performance. As these bounds depend on the kernel k and on C one can attempt to choose these parameters so as to minimize the generalization

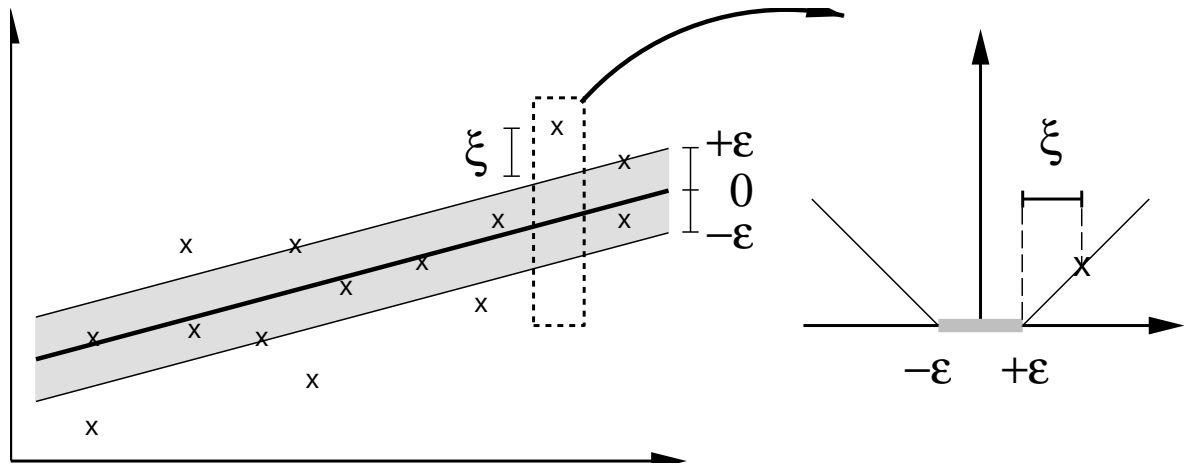

Figure 4: Left: Fitting a straight line to data using the ϵ-insensitive loss function. Right: The ϵ-insensitive loss function.

error. In practice these bounds are typically rather loose. An alternative approach is to use cross-validation to choose these parameters.

3 Support Vector Regression

The previous two sections have concentrated on classification problems. However, similar ideas can be applied to the regression problem. Suppose that we want to predict a real-valued target. Let us start with a linear regression problem; we simply remove the sgn function and let our prediction be expressed as

$$f(\mathbf{x}) = \tilde{\mathbf{w}}.\mathbf{x} + w_0, \tag{14}$$

(compare this to equation 5). If we use the standard squared error measure $E_{sq} = \frac{1}{2}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2$ then we can obtain $\tilde{\mathbf{w}}$ by inverting $(D+1)\times(D+1)$ matrix, where D is the dimension of $\mathbf{x}$.

However, if we use a different error function, the ϵ-insensitive error function, then a sparse solution emerges. The ϵ-insensitive error function is defined as

$$E_\epsilon(z) = \begin{cases} |z| - \epsilon & \text{if } |z| \geq \epsilon, \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

This function is plotted in Figure 4(right). By minimizing the error function $C\sum_{i=1}^{n} E_\epsilon(y_i - f(\mathbf{x}_i))$ we obtain a solution $\tilde{\mathbf{w}} = \sum_{i=1}^{n}\beta_i\mathbf{x}_i$, where many of the coefficients β_i are zero. The data points which lie inside the ϵ-"tube" have $\beta_i = 0$, those on the edge or outside have non-zero β_i.

This problem can again be kernelized, so that our prediction $y(\mathbf{x})$ is expressed as

$$f(\mathbf{x}) = \sum_{i=1}^{n}\beta_i k(\mathbf{x}, \mathbf{x}_i) + w_0, \tag{16}$$

(compare this to equation 13). The problem is now to determine the coefficients $\{\beta_i\}$. If we use the standard squared error measure $E_{sq} = \frac{1}{2}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2$ then we can obtain $\boldsymbol{\beta}$ by inverting an $n \times n$ matrix. In this case it is very likely that all n of the β_i's will be non-zero. Again, if the ϵ-insensitive error function is used, then a sparse solution emerges. We minimize an objective function that trades off the error $C\sum_{i=1}^{n} E_\epsilon(y_i - f(\mathbf{x}_i))$ against the length of the weight vector in feature space $||\tilde{\mathbf{w}}||^2$. The solution for $\boldsymbol{\beta}$ can again be found by solving a quadratic programming problem. The regression function using this solution is known as support vector regression or SVR. To use SVR in practice it is necessary to set both C and ϵ and the kernel. Again this can be approached by optimizing VC bounds or by cross-validation approaches.

4 Comparison of SVMs with linear and logistic regression

Comparing the SVM for classification with logistic regression, we see that in fact the differences are not that large, but the main difference is the *sparsity* of the SVM solution. Both methods build a classifier that is linear in $\mathbf{x}$. Although the SVM can make use of the kernel trick, in fact the same trick can also be used to produce what is called "kernel logistic regression".

For regression, we again see that the differences between linear regression and SVR are not that large, except for the *sparsity* of the SVR solution. Again the "kernel trick" can be applied to linear regression as well as to SVR; a fully probabilistic version of this method is known as Gaussian process regression, see [7].

Although the SVM has been applied very successfully to many problems, notice that it is a "shallow" architecture with only one hidden layer. Bengio and Le Cun [2] *inter alia* have argued that to tackle harder AI problems it will be necessary to use "deeper" networks that have multiple layers of non-linear processing.

5 SVMs in practice

SVMs have been applied to a wide variety of problems; here we highlight two examples. The first of these is the problem of isolated handwritten digit recognition. This problem is of interest to the US Postal Service (e.g. for automated ZIP code reading) and is widely used as a real-life classification example. Results in [8, sec. 7.8.1]. show that for polynomial, Gaussian and tanh kernels, error rates of around 4% were obtained. This is similar (or slightly better) than other approaches on this dataset, some of which were tuned using specialist knowledge of the problem. Also it is of note that only $\sim 4\%$ of the 7291 training examples were support vectors.

Our second example of the use of SVMs is in text classification. For example the news agency Reuters has collected 21450 news stories and indexed them into 135 different categories. The features typically used to classify these documents are obtained from word frequencies within a document (this is sometimes called the "bag of words" representation of the document, as it competely ignores the order in which the words appear in the

document). Joachims [6] has applied SVMs to this problem and obtained excellent results.

References

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

[2] Y. Bengio and Y. Le Cun. Scaling learning algorithms towards ai. In L. Bottou, O. Chapelle, D. DeCoset, and J. Weston, editors, *Large Scale Kernel Machines*, pages 321–359. MIT Press, 2007.

[3] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):1–47, 1998.

[4] C. Cortes and V. Vapnik. Support Vector networks. *Machine Learning*, 20:273–297, 1995.

[5] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

[6] T. Joachims. Text categorization with support vector machines. In *European Conference on Machine Learning (ECML)*, 1998.

[7] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, Massachusetts, 2006.

[8] B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, 2002.