Support Vector Machines

April 2, 2017

Contents

1. Introduction

The idea is to build a classifier which first makes an (optional) transformation of the input data, and then a linear separation where the decision boundary is placed to give maximal margins to the available data points. The location of the decision boundary is given by the weights ($\vec{w}$) and the bias (b) so the problem is to find the values for $\vec{w}$ and b which maximizes the margin, i.e. the distance to any datapoint.
The *primal* formulation of this optimization problem can be stated mathematically like this:

$$\min_{\vec{w},b} ||\vec{w}|| \tag{1}$$

under the constraints

$$t_i(\vec{w}^T \cdot \phi(\vec{x}_i) + b) \geq 1 \qquad \forall i \tag{2}$$

where we have used the following notation:

$\vec{w}$	Weight vector defining the separating hyperplane
b	Bias for the hyperplane
$\vec{x}_i$	The ith datapoint
t_i	Target class (-1 or 1) for datapoint i
$\phi(\ldots)$	Optional transformation of the input data

The constraints (2) enforce that all datapoints are not only correctly classified, but also that they stay clear of the decision boundary by a certain margin. Solving this optimization problem results in values for $\vec{w}$ and b which makes it possible to classify a new datapoint $\vec{x}^\star$ using this *indicator* function:

$$\text{ind}(\vec{x}^\star) = \vec{w}^T \cdot \phi(\vec{x}^\star) + b \tag{3}$$

If the indicator returns a positive value, we say that $\vec{x}^\star$ belongs to class 1, if it gives a negative value, we conclude that the class is -1. All the training data should have indicator values above 1 or below -1, since the interval between -1 and 1 constitutes the margin.
The bias variable, b, can be eliminated by a standard trick, i.e. by incorporating it as an extra element in the weight vector $\vec{w}$. We then need to let the $\phi(\ldots)$ function append a corresponding constant component, typically the value 1. Note that by using this trick we are actually slightly modifying the problem, since we are now also including the bias value in the cost function ($||\vec{w}||$). In practice, this will not make much difference, and we will use this "bias free" version from here on.

2. Dual Formulation

The optimization problem can be transformed into a different form, called the *dual problem* which has some computational advantages. In particular, it makes it possible to use the *kernel trick*, thereby eliminating the need for evaluating the $\phi(\ldots)$ function directly. This allows us to use transformations into very high-dimensional spaces without the penalty of excessive computational costs.
The dual form of the problem is to find the values α_i which minimizes:

$$\frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j t_i t_j \mathcal{K}(\vec{x}_i, \vec{x}_j) - \sum_i \alpha_i \tag{4}$$

subject to the constraints

$$\alpha_i \geq 0 \quad \forall i \tag{5}$$

The function $\mathcal{K}(\vec{x}_i, \vec{x}_j)$ is called a *kernel function* and computes the scalar value corresponding to $\phi(\vec{x}_i) \cdot \phi(\vec{x}_j)$. This is, however, normally done implicitly, i.e., without actually computing the two vectors and taking their scalar product .
The indicator function now takes the form:

$$\text{ind}(\vec{x}^\star) = \sum_i \alpha_i t_i \mathcal{K}(\vec{x}^\star, \vec{x}_i) \tag{6}$$

For normal data sets, only a handful of the will be non-zero. Most of the terms in the indicator function will therefore be zero, and since this is known beforehand its evaluation can often be made very efficient.
This formulation has some interesting facts to be pointed out. First, the coefficients α's are nothing else than the Langrange multipliers. Second, only the multipliers associated to the support vectors are non zero. Most of the terms in the indicator function will therefore be zero, and since this is known beforehand its evaluation can often be made very efficient. This backs one of the most important characteristics of SVM; the fact that the decision boundary is affected only by support vectors. The hyperplane is not sensitive to any point correctly classified outside of the margin. Third, it naturally opens the door to kernels.

3. Matrix Formulation

The dual problem can be expressed in a more compact form using vectors and matrices: find the vector $\vec{\alpha}$ which minimizes

$$\frac{1}{2}\vec{\alpha}^T P \vec{\alpha} - \vec{\alpha} \cdot \vec{1} \qquad \text{where } \vec{\alpha} \geq \vec{0} \tag{7}$$

$\vec{1}$ denotes a vector where all elements are one. Correspondingly, $\vec{0}$ is a vector with all zeros. We have also introduced the matrix P, with these elements:

$$P_{i,j} = t_i t_j \mathcal{K}(\vec{x}_i, \vec{x}_j) \tag{8}$$

In fact, this is a standard form for formulating quadratic optimization problems with linear constraints. This is a well known class of optimization problems where efficient solving algorithms are available.

4. Adding Slack Variables

The above method will fail if the training data are not linearly separable. In many cases, especially when the data contain some sort of noise, it is desirable to allow a few datapoints to be misclassified if it results in a substantially wider margin. This is where the method of *slack variables* comes in.
Instead of requiring that *every* datapoint is on the right side of the margin (equation 2) we will now allow for mistakes, quantified by variables ξ_i (one for each datapoint). These are called *slack variables*. The constraints will now be

$$t_i(\vec{w}^T \cdot \phi(\vec{x}_i)) \geq 1 - \xi_i \qquad \forall i \tag{9}$$

(Remember that we have already eliminated b from (2) by including it as a weight).
To make sense, we must ensure that the slack variables do not become unnecessarily large. This is easily achieved by adding a penalty term to the cost function, such that large ξ values will be penalized:

$$\min_{\vec{w},\vec{\xi}} ||\vec{w}|| + C \sum_i \xi_i \tag{10}$$

The new parameter C sets the relative importance of avoiding slack versus getting a wider margin. This has to be selected by the user, based on the character of the data. Noisy data typically deserve a low C value, allowing for more slack, since individual datapoints in strange locations should not be taken too seriously.
Fortunately, the dual formulation of the problem need only a slight modification to incorporate the slack variables. In fact, we only need to add an extra set of constraints to (5):

$$\alpha_i \geq 0 \quad \forall i \quad \text{and} \quad \alpha_i \leq C \quad \forall i \tag{11}$$

Equation (4) stays the same.

5. Selection of Kernel Function

What if we simply cannot find a "linear" hyperplane to separate our classes? In this case it is natural to try moving beyond linearity. But how? A good idea could be to apply a transformation to our x , remapping the features to a wider (or just different) space where the classes are separable. For instance, it is not hard to show that two classes concentrically split in a cartesian space are linearly separable in polar coordinates(Figure. 1).

By transforming the input data non-linearly to a high-dimensional space, more complex decision boundaries can be utilized. In the dual formulation, these transformed data points $\phi(\vec{x}_i)$ always appear in pairs, and the only thing needed is the scalar product between the pair. This makes it possible to use what is often referred to as the *kernel trick*, i.e. we do not actually have to make the data transformation but, instead, we use a kernel function which directly returns the scalar product $\phi(\vec{x}_i) \cdot \phi(\vec{x}_j)$.
Here are the most commonly used kernel functions:

- Linear kernel

$$\mathcal{K}(\vec{x}, \vec{y}) = \vec{x}^T \cdot \vec{y} + 1$$

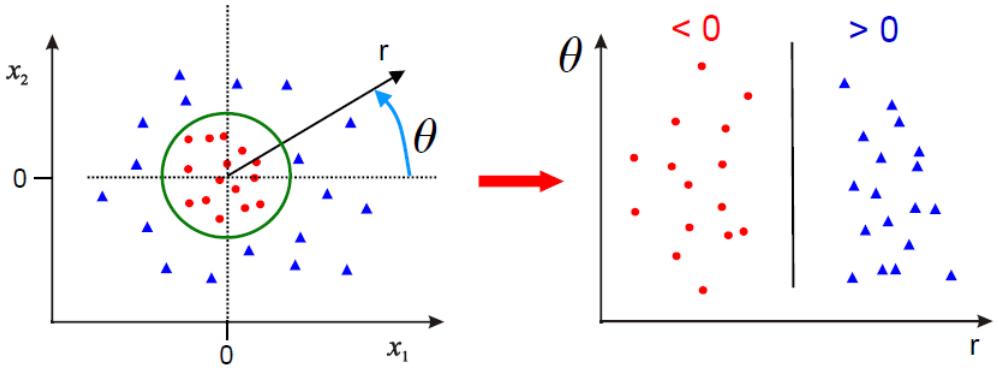

Figure 1: Kernel trick for SVMs

This kernel simply returns the scalar product between the two points. This results in a linear separation. Note the addition of 1 which comes from the elimination of the bias (b) by appending a constant element 1 to the data, resulting in an extra term 1×1 added to the scalar product.

- Polynomial kernels

$$\mathcal{K}(\vec{x}, \vec{y}) = (\vec{x}^T \cdot \vec{y} + 1)^p$$

This kernel allows for curved decision boundaries. The exponent p (a positive integer) controls the degree of the polynomials. $p = 2$ will make quadratic shapes (ellipses, parabolas, hyperbolas). Setting $p = 3$ or higher will result in more complex shapes.

- Radial Basis Function kernels

$$\mathcal{K}(\vec{x}, \vec{y}) = e^{-\frac{(\vec{x}-\vec{y})^2}{2\sigma^2}}$$

This kernel uses the explicit difference between the two datapoints, and often results in very good boundaries. The parameter σ can be used to control the smoothness of the boundary.

- Sigmoid kernels

$$\mathcal{K}(\vec{x}, \vec{y}) = \tanh(k\vec{x}^T \cdot \vec{y} - \delta)$$

This is yet another possible non-linear kernel. Parameters k and δ need to be tuned to get best performance.