Study Notes for AI (CS 520)
Lectures 24-25: Support Vector Machines

Corresponding Book Chapters: Chapters 18.8-18.9
Note: These notes provide only a short summary and some highlights of the material covered in the corresponding lecture based on notes collected from students. Make sure you check the corresponding chapters. Please report any mistakes in or any other issues with the notes to the instructor.

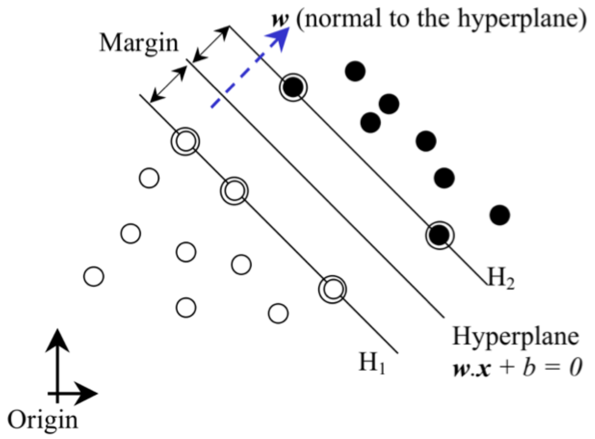
1 Support Vector Machines

Other classifiers assumed that the class boundaries are easily determined by linear functions or a multi-dimensional Gaussian function. The later assumption is justified by the common occurrence of the Gaussian distribution in nature, but there are clearly classes for which the boundary will not be Gaussian. The techniques available for these cases are mixture models, usually involving the sum of several Gaussians, or piecewise linear solutions such as neural networks. These non-linear techniques are computationally expensive involving some optimization to achieve good performance.

Support vector machines are the most recent non-linear classifier to appear. Vapnik, who originally proposed support vectors took a radically different approach to classical methods:

"In contrast to classical methods of statistics where in order to control performance one decreases the dimensionality of a feature space, the Support Vector Machine (SVM) dramatically increases dimensionality and relies on the so-called large margin factor."

A support vector machine is primarily a two-class classifier. It is possible to solve multi-class problems with support vectors by treating each single class as a separate problem. The objective is to maximize the width of the margin between classes, that is, the empty area between the decision boundary and the nearest training patterns.


Margin
$\boldsymbol{w}$ (normal to the hyperplane)
H_2
Hyperplane
$\boldsymbol{w}.\boldsymbol{x} + b = 0$
H_1
Origin

Let us consider the case where the class boundary is a hyperplane. Given a set of points x_i in n-dimensional space with corresponding classes $\{y_i : y_i \in \{-1, 1\}\}$ then the training algorithm attempts to place an hyperplane between points where $y_i = 1$ and points where $y_i = -1$. Once this has been achieved a new pattern x can then be classified by testing which side of the hyper-plane the point lies on.

In the simplest case, linear machines are trained on separable data. The training data is given by $(x_1, y_1), \ldots, (x_N, y_N)$ where each x_i is a vector of real numbers and the corresponding y_i designates the class of x_i,i.e., as 1 or 1. Suppose the two classes can be separated by a hyperplane:

$$(wx) + b = 0.$$

We say that this set is separated by the optimal hyperplane if it is separated without error and the distance between the closest points and the hyperplane is maximal. The training points, which lie on one of the hyperplanes (H_1, H_2) and whose removal would change the solution found are called support vectors.

To describe the separating hyperplane let us suppose that all the training data satisfy the following constraints:

$$(w \cdot x_i) + b \geq +1 \text{ for } y_i = +1$$
$$(w \cdot x_i) + b \leq 1 \text{ for } y_i = -1$$

which can be combined into one set of inequalities:

$$y_i(w \cdot x_i + b)1 \geq 0 \text{ for } i = 1 \ldots N$$

Now consider the points for which the equality in the previous equations holds. These points lie on one of two planes:

$$H_1 : (w \cdot x_i) + b = 1 \quad \text{or} \quad H_2 : (w \cdot x_i) + b = 1$$

with normal w and perpendicular distance from the origin respectively $\frac{|1b|}{|w|}$ and $\frac{|1b|}{|w|}$. The shortest distance between H_1 and H_2 is the difference, $\frac{2}{|w|}$, and the distance between the separating hyperplane and the closest points is $\frac{1}{|w|}$.

1.1 Finding the separating hyperplane

The distance separating the classes is $\frac{2}{|w|}$, the optimal hyperplane therefore is the one for which $|w|$ is minimal. Finding the hyperplane is formulated as the following optimization problem:

$$\text{Minimize } |w|^2 \text{ subject to constraints } \forall(x_i, y_i) : y_i(w \cdot xi + b) - 1 \geq 0.$$

We will not consider in depth the solution to this optimization problem, but it is worth making some observations about it. In particular, the optimal solution is found by solving the following problem:

$$\arg\max_{\alpha} \sum_j \alpha_j - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k (x_j \cdot x_k)$$

subject to the constraints $a_j \geq 0$ and $\sum_j \alpha_j y_j = 0$. This is a quadratic programming optimization problem, for which there are good software packages. Once we have found the vector α, we can then compute the margin w with the equation: $w = \sum_j \alpha_j x_j$.

There are three important properties of the above expression. First, it is convex, i.e., it has a single global optimum that can be found efficiently. Second, the data enter the expression only in the form of dot products of pairs of points $(x_j \cdot x_k)$. This is an important property that will allow us to generalize the method to the non-linear case in a computationally efficient manner. A final important property is that the weights α_j associated with each data point are zero except for "support vectors", i.e., the points closest to the separating hyperplane. There are usually many fewer support vectors than examples, which provides computational benefits to SVMs.

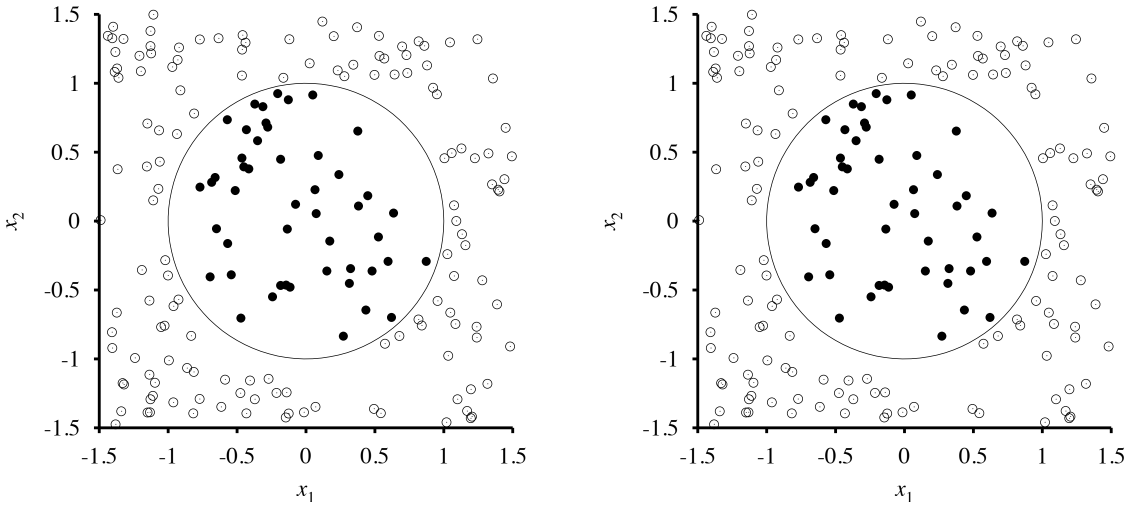
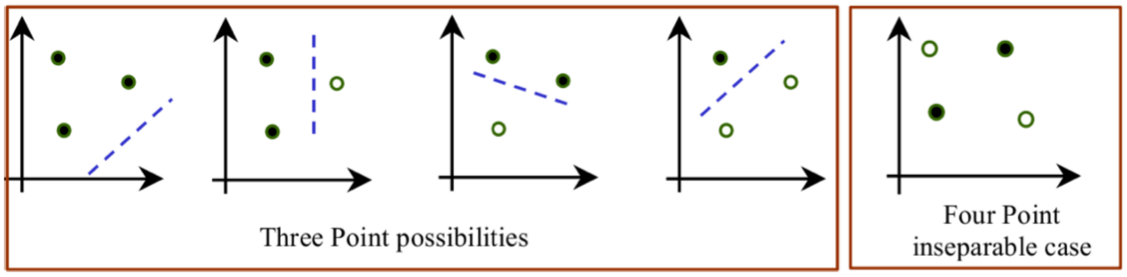


Figure 1: (left) An example with 2D training data, which cannot be linearly separated. (right) The same data are mapped to a 3D space $x_1^2, x_2^2, \sqrt{2}x_1x_2)$. The circular decision boundary in the original space becomes a linear decision boundary in three dimensions.

2 Non-Linear Support Vector Machines

What if the examples are not linearly separable? Figure 2(left) shows an input space that cannot be separated by a line. The non-linear SVM is based on the following idea. We map the points in the data set to a higher dimensional space where it is possible to separate them with a linear hyperplane. The mapping is non-linear and is achieved by a suitable kernel function. To demonstrate that the idea always works we first introduce a measure named the Vapnik and Chervonenkis (VC) dimension:

The VC dimension of a class of functions f_i is the maximum number of points that can be separated (shattered) into two classes in all possible ways. We illustrate this using a simple example. Given the class f corresponds to straight lines on a 2D plane, then any three (non collinear) points can be separated. The following figure shows all the possible ways in which three non collinear points can be divided into two classes.



However for four points there is the case that they cannot be separated by a single line, thus the VC dimension of a line is 3. If we extend the dimension to three then all possible four point cases can be separated by a hyperplane.

This generalizes to a fundamental theorem:

The VC dimension of the set of oriented hyperplanes in $\mathbb{R}^n$ is $n+1$

which implies that it is always possible, for a finite point set, to find a dimension where all possible separations of the point set can be achieved by a hyperplane. If we map our points to a space of sufficiently high dimension we can use the linear SVM training algorithm described above to find the optimal separating hyperplane. Suppose that function ϕ maps points from an n-dimensional space to an m-dimensional space where $m > n$:

$$\phi : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

Then, by replacing $x_i \cdot x_j$ by $\phi(x_i) \cdot \phi(x_j)$ in all the equations of the Lagrange optimization formulation we can find the optimal separating hyperplane normal vector w in R^m. However, mapping a point into a higher dimensional space might well involve heavy computation time and storage requirements, so in practice we use a kernel function that directly computes the dot product in the higher dimensional space.

$$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$$

Having chosen an appropriate kernel we replace every $x_i \cdot x_j$ with $K(x_i, x_j)$ everywhere in the optimization algorithm, and thus avoid any need to carry out the mapping explicitly. Using a kernel, the training procedure estimates w in the low dimension space, but the membership test is done on the sign of $K(w, x) + b$ rather than $(w \cdot x + b)$, which was used for the linear case.

2.1 Example

Suppose that our vectors are in $\mathbb{R}^2$ and we choose:

$$K(u, v) = (uv)^2 = u_1^2 v_1^2 + 2u_1 v_1 u_2 v_2 + u_2 v_2$$

It is easy to find a mapping ϕ from R^2 to R^3 such that:

$$(u \cdot v)^2 = \phi(u) \cdot \phi(v)$$

Neither the mapping ϕ nor the dimension of the new space are unique. Some possibilities are, for example:

$$\phi(x) = (x_1^2 \quad \sqrt(2) x_1 x_2 \quad x_2^2)$$

or:

$$\phi(x) = \frac{1}{\sqrt{2}}(x_1^2 - x_2^2 \quad 2x_1 x_2 \quad x_1^2 + x_2^2)$$

or, lifting the dimension to 4:

$$\phi(x) = (x_1^2 \quad x_1 x_2 \quad x_1 x_2 \quad x_2^2)$$

2.2 Mercer's condition

Since we don't need the lifting function in the algorithm, we do not need to make a selection of ϕ, all we need is the kernel function K. Nevertheless, in order to make the technique work we need to ensure that there exists an appropriate lifting function. One test for this is known as Mercer's condition.

If K is a function on vectors u and v then there exists a mapping ϕ into some vector space with an expansion:

$$K(u, v) = \phi(u) \cdot \phi(v)$$

if and only if for any $g(u)$ such that:

$$\int g(u)^2 du$$

is finite then:

$$\int K(u,v)g(u)g(v)dudv \geq 0$$

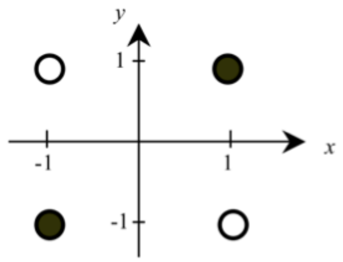
2.3 Some popular non-linear kernels

The most commonly used kernel functions that obey Mercer's condition are:

- Polynomial: $K(u,v) = (u \cdot v + 1)^p$
- Gaussian Radial Basis Function: $K(u,v) = exp(\frac{-||u-v||^2}{2\sigma^2})$
- Sigmoidal (hyperbolic separating surface): $K(u,v) = tanh(ku \cdot v - \delta)$

The radial basis kernel is interesting because it implies that the corresponding lifting function ϕ will raise the dimension of the data space to infinity. This can be seen by that fact that there is an infinite series expansion of e^x. This is needed to gather the dot product terms that equate with the exponential function. Thus, the radial basis function can solve any classification problem regardless of size. It also behaves in a predictable (though not always ideal) way.

2.4 Example: SVM for the XOR problem



The exclusive OR is the simplest problem that cannot be solved using a single linear function operating on the features. We will lift the data from the 2 dimensional space to a six dimensional space using a quadratic polynomial kernel:

$$K(u,v) = (u \cdot v + 1)^2$$

For our two dimensional problem we can write the kernel as:

$$K(u,v) = (u_1v_1 + u_2v_2 + 1)^2 = (u_1v_1)^2 + (u_2v_2)^2 + 2u_1v_1u_2v_2 + 2u_1v_1 + 2u_2v_2 + 1$$

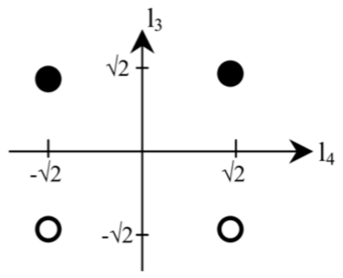
Mercer's condition holds for all polynomial kernels, so we are guaranteed a lifting function exists. We don't need to find one, but for example:

$$\phi(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1)$$

is one possibility. You can easily verify that it satisfies $K(u,v) = \phi(u) \cdot \phi(v)$. The new space has dimension 6, and, for the sake of example, we calculate the new six dimensional coordinates corresponding to our four original points:

x_1	x_2	$l_1 = x_1^2$	$l_2 = x_2^2$	$l_3 = \sqrt{2}x_1x_2$	$l_4 = \sqrt{2}x_1$	$l_5 = \sqrt{2}x_2$	$l_6 = 1$
1	1	1	1	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	1
-1	-1	1	1	$\sqrt{2}$	$-\sqrt{2}$	$-\sqrt{2}$	1
1	-1	1	1	$-\sqrt{2}$	$\sqrt{2}$	$-\sqrt{2}$	1
-1	1	1	1	$-\sqrt{2}$	$-\sqrt{2}$	$\sqrt{2}$	1

It can be seen that the points are separable on the l_3 axis. It turns out that a plane, which is perpendicular to the l_3 axis and goes through the origin, properly separates the mapped points. Remember that in the actual algorithm we do not calculate (or even need to know) the lifting function ϕ.



2.5 Final Comment

An important benefit of the support vector machine approach is that the complexity of the resulting classifier is characterized by the number of support vectors rather than the dimensionality of the transformed space. As a result, support vector machines tend to be less prone to problems of over-fitting than some other methods.