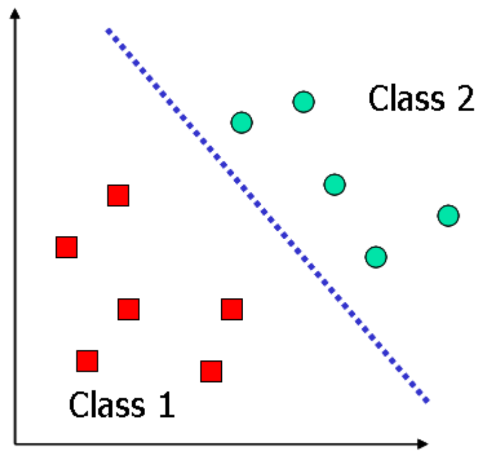
A Brief Introduction to Support Vector Machines

Support vector machines (SVM) are a relatively new technique in machine learning. Today they are probably the hottest technique out there, eclipsing neural networks and perhaps genetic algorithms. They were invented by Vladimir Vapnik in 1992 from a theoretical perspective, and have only begun to be used in machine learning since about 1999. They are most often compared to neural networks, because both are often equivalent (in terms of what kind of function they can learn) when configured appropriately. In fact, a perceptron algorithm can be used to train a SVM.

However, SVM's have a number of much nicer properties than neural networks, which includes generally much faster training, easier configuration, and nicer theoretical properties. For example, Bottou compared SVM's with neural networks on a handwriting recognition task and found an error rate of 1.1% which was identical to that of a carefully constructed neural network.

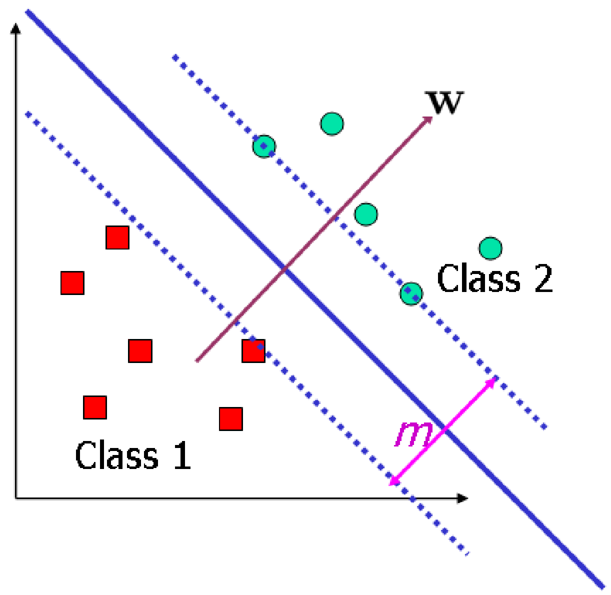
In this lecture we will only give a general introduction to SVM's and leave out many of the mathematical details.

First, consider a classification problem with two features. Each is a numeric feature labeled on the X and Y axis:



With a linear discriminant we want to find a line that separates the two classes, and we've discussed numerous ways to do this (decision trees, nearest neighbor, neural network, etc.) Let's say we find such a line. Are all decision boundaries the same as long as they separate the classes?

The concept behind SVM's is no, not all decision boundaries are equal. There is one that is better than the rest, namely the boundary that is as far from both classes as possible:



The dark blue line in the middle is the separating boundary that attempts to maximize the distance between the two classes. We can draw dashed lines parallel to this separating boundary where at least one data point from each class is on the line, if we are truly maximizing the distance. Samples on these lines are called the support vectors (3 in the above figure). This sounds good, but how do you find such a line?

Algebraically, the idea is to maximize the margin *m*, which is the shortest distance between the dashed lines. Note that in the more general case, with an n-dimensional space, these lines would be hyper planes.

Let *w* be a vector perpendicular to the separating the dashed lines, and hence, parallel to *m*. Our decision function to determine whether or some point x in the space is in one class or another is using:

$$w \bullet x + b \geq 0$$

The point x is described as a vector, and b is some constant (used to increase the margin).

Now we add some additional constraints for our training data. For everything in class 2 we will insist that:

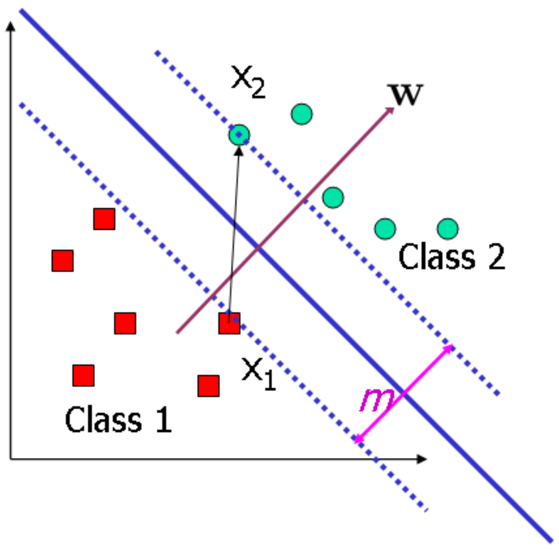
$$w \bullet x + b \geq 1$$

And for everything in class 1 we will insist that:

$$w \bullet x + b \leq -1$$

In the gap between the two dashed vectors we get values from our decision function that range from -1 to +1 but all of the training data must be <= -1 or >= 1 after computing our decision function. These constraints must now be used to compute the vector w in such a way as to maximize the margin *m*.

Let's label some of these support vectors on our space:



What we would like to do is maximize *m*. How do we compute it? Note that X_2 falls on the line with a function value of 1, while X_1 falls on the line with a function value of -1.

If we compute: w / | w | then this gives us a unit vector in the direction of w. Since w is parallel to m, we can find m by computing the dot product of (x2 – x1) (recall that the dot product with a unit vector is the projection down to the unit vector's direction, i.e. it is |v1||v2|cosΘ). This gives us:

(1) $$m = \frac{w}{\|w\|} \bullet (x_2 - x_1)$$

For the support vectors, x2 and x1, we have that:

$$w \bullet x_2 + b = 1$$
$$w \bullet x_1 + b = -1$$

If we subtract one from the other the b's subtract out and we end up with:

(2) $$w \bullet (x_2 - x_1) = 2$$

If we plug this into (1) then we end up with:

(3) $$m = \frac{w}{\|w\|} \bullet (x_2 - x_1) = \frac{2}{\|w\|}$$

If I want the widest possible gap, this means we need to minimize |w| subject to our inequality constraints.

If we reach into our bag of mathematical tricks, we can solve this using Lagrange multipliers. There are other techniques as well, but Lagrange multipliers are the most elegant, mathematically. There are some extra details though, because Lagrange multipliers depends on equality constraints, not inequalities.

After going through the hairy math, we would end up with that w is computed using the sum of the training points:

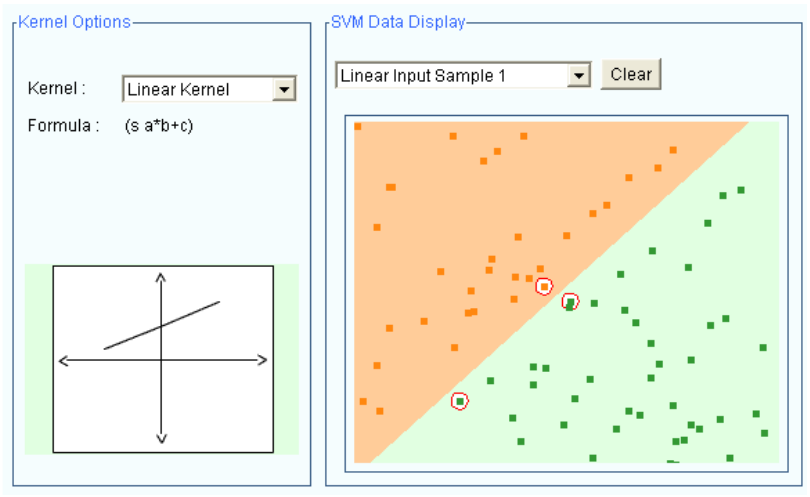
$$W = \sum_i C_i x_i$$

Where C_i is some constant and the x's are the training samples.

The next question is how do you find the C's? One technique is to use hill climbing. The beautiful property of our problem space is such that it is convex, so we don't need to worry anymore about local maxima. Hill climbing will simply converge to the optimal answer, unlike an ordinary neural network.

How computationally tractable is this? To compute the C's we'll need to compute dot products of pairs of vectors in the training space, which can be computationally challenging but tractable. Most of these will end up being zero (the ones we really need are just the support vectors along the gap).

Demo using a linear function:
http://www.eee.metu.edu.tr/~alatan/Courses/Demo/AppletSVM.html



At this point you might be saying, "OK that's neat, but what good is this? We have lots of problems that are not linearly separable."

The answer to this problem is YES. To solve it there is an additional step that is applied to SVM's. The input space is mapped to a separate space using a **kernel** function. This is a similar idea to fourier transforms (converting a signal from the time domain to the frequency domain) or what you might have done using log-scale plots.

To make the data linearly separable we could:

1. Project the data from the input space to a new space called "feature space"
2. This feature space may have more dimensions than the input space so we could separate the data linearly in the feature space
3. Use the normal linear SVM in this feature space

Here is an example projection:

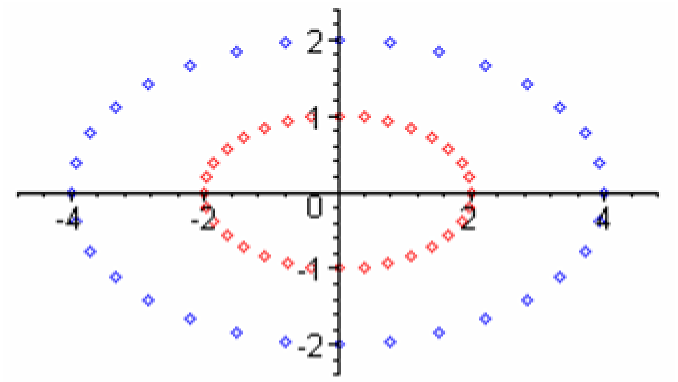
$$P : I \Rightarrow F$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_1 x_2 \\ x_2^2 \end{pmatrix}$$

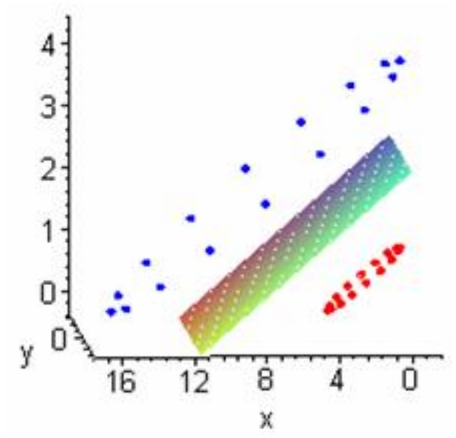
The inner product of two vectors x and y projected in the space F becomes:

$$\begin{pmatrix} x_1^2 \\ x_1 x_2 \\ x_2^2 \end{pmatrix} \cdot \begin{pmatrix} y_1^2 \\ y_1 y_2 \\ y_2^2 \end{pmatrix} = x_1^2 y_1^2 + x_1 x_2 y_1 y_2 + x_2^2 y_2^2$$

Here is a picture of what this can do. Here is input in the input space that is not linearly separable:



After we apply the kernel function we end up with something like:



This new version is linearly separable! After we solve the problem by finding the separating hyperplane in F we transform it back to the original input space I.

For efficiency purposes, we don't have to actually go through all the work of projecting each vector to the new feature space. Since we only care about dot products in the feature space, we can compute it implicitly by squaring the dot products in the input space. In our example:

$$\langle \vec{x}, \vec{y} \rangle^2 = \left[\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \right]^2 = (x_1 y_1 + x_2 y_2)^2 = x_1^2 y_1^2 + x_1 x_2 y_1 y_2 + x_2^2 y_2^2$$

$$\begin{pmatrix} x_1^2 \\ x_1 x_2 \\ x_2^2 \end{pmatrix} \cdot \begin{pmatrix} y_1^2 \\ y_1 y_2 \\ y_2^2 \end{pmatrix} = x_1^2 y_1^2 + x_1 x_2 y_1 y_2 + x_2^2 y_2^2$$

This is the magic of calculating support vectors that makes it tractable.

Common kernel functions are the polynomial and radial basis functions.

Polynomial:

$$k(\vec{x}, \vec{y}) = (\langle \vec{x}, \vec{y} \rangle + b)^n$$

Here, n is its order (in our example n=2) and b is called the lower order term (in our example b=0).

Gaussian radial basis function:

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

This is a powerful kernel as its effect is to create a small classification “hyperball” around a training sample. This kernel doesn’t have a projection formula since its dimension is infinite (you can create as many “balls” as you want). Theta is a measure of the radius of the “hyperball” around an instance. You want this ball to be big enough so “hyperballs” connect with each other but not too big to overlap the other class.

If the hyperballs don’t connect then you’ll overfit or memorize the training data.

Demo: Run applet with polynomial and radial basis functions, with different values for theta.

In this lecture we have only given an overview of how SVM’s work. For the mathematically inclined, you’re welcome to investigate further. There are many excellent tutorials online. For the non-mathematically inclined, as long as you understand the major concepts of what is going on, this is sufficient for you to use SVM’s as a tool in your chest of machine learning techniques.