

## Support Vector Machines

“A machine with too much capacity is like a botanist with a photographic memory who, when presented with a new tree, concludes that it is not a tree because it has a different number of leaves from anything she has seen before; a machine with too little capacity is like the botanist’s lazy brother, who declares that if it’s green, it’s a tree. Neither can generalize well.” (C. Burges, *A tutorial on support vector machines for pattern recognition*).

“The exploration and formalization of these concepts has resulted in one of the shining peaks of the theory of statistical learning.” (Ibid.)

That “shining peak” is the 1979 paper by V. Vapnik, “Estimation of dependences based on empirical data” (in Russian) defining what is now called “VC dimension”.

These ideas (and others) were fleshed out more thoroughly in Vapnik’s 1995 book, *The Nature of Statistical Learning Theory*.

This is about the time that support vector machines became a very popular topic of research in the machine learning community.

## Support Vector Machines (Vapnik, 1979)

- Assume a linearly separable binary classification problem.

- Instances are represented by vector  $\mathbf{x} \in \mathfrak{R}^n$ .

- Training examples:

$$S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m) \mid (\mathbf{x}_i, y_i) \in \mathfrak{R}^n \times \{+1, -1\}\}$$

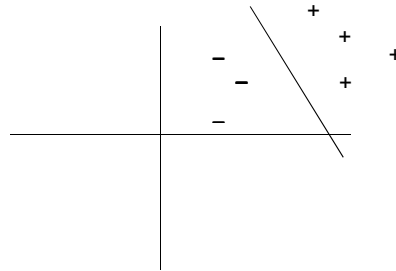
- Hypothesis: A function  $h: \mathfrak{R}^n \rightarrow \{+1, -1\}$ .

$$h(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + w_0) = \text{sgn}\left(\sum_{i=1}^n w_i x_i + w_0\right)$$

$$\text{where } \text{sgn}(z) = \begin{cases} 1 & \text{if } z > 0 \\ -1 & \text{otherwise} \end{cases}$$

- Positive and negative instances are to be separated by the hyperplane

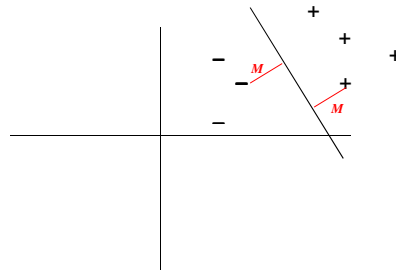
$$\mathbf{w} \cdot \mathbf{x} + w_0 = 0$$



- Intuition: the best hyperplane (for future generalization) will “maximally” separate the examples

- Positive and negative instances are to be separated by the hyperplane

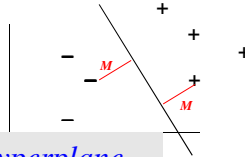
$$\mathbf{w} \cdot \mathbf{x} + w_0 = 0$$



- Intuition: the best hyperplane (for future generalization) will “maximally” separate the examples
- Assume that the hyperplane  $h$  is placed so that the perpendicular distance  $M$  to the closest positive training example equals the perpendicular distance to the closest negative training example.  $M$  is called the *margin* of the training set with respect to the hyperplane.

- Positive and negative instances are to be separated by the hyperplane

$$\mathbf{w} \cdot \mathbf{x} + w_0 = 0$$



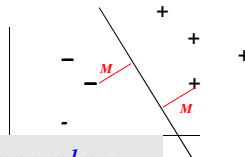
*Vapnik showed that the hyperplane maximizing the margin of  $S$  will have minimal VC dimension in the set of all consistent hyperplanes, and will thus be optimal.*

generalization) will

- Assume that the hyperplane  $h$  is placed so that the perpendicular distance  $M$  to the closest positive training example equals the perpendicular distance to the closest negative training example.  $M$  is called the **margin** of the training set with respect to the hyperplane.

- Positive and negative instances are to be separated by the hyperplane

$$\mathbf{w} \cdot \mathbf{x} + w_0 = 0$$



*Vapnik showed that the hyperplane maximizing the margin of  $S$  will have minimal VC dimension in the set of all consistent hyperplanes, and will thus be optimal.*

generalization) will

**This is an optimization problem!**

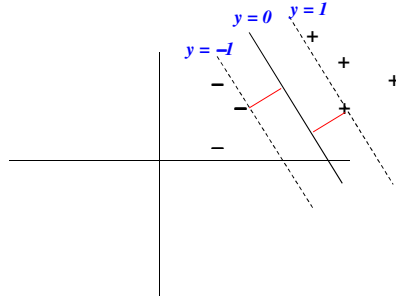
- Assume that the hyperplane  $h$  is placed so that the perpendicular distance  $M$  to the closest positive training example equals the perpendicular distance to the closest negative training example.  $M$  is called the **margin** of the training set with respect to the hyperplane.

## Geometry of the Margin

We can scale  $\mathbf{w}$  and  $w_0$  such that:

$$y = \mathbf{x}_i \cdot \mathbf{w} + w_0 \geq +1 \text{ for all positive instances } (y_i = +1)$$

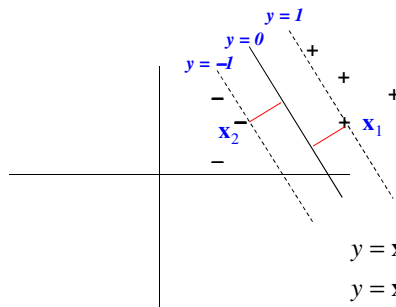
$$y = \mathbf{x}_i \cdot \mathbf{w} + w_0 \leq -1 \text{ for all negative instances } (y_i = -1)$$



In this case, the margin  $M = \frac{1}{\|\mathbf{w}\|} \equiv \frac{1}{\sqrt{\mathbf{w} \cdot \mathbf{w}}}$

**Proof:** Suppose  $\mathbf{x}_1$  is a positive example on the margin, and  $\mathbf{x}_2$  is a negative example on the margin. Then

$$\begin{aligned} \mathbf{x}_1 \cdot \mathbf{w} + b &= +1 & \text{and} & & \mathbf{w}(\mathbf{x}_1 - \mathbf{x}_2) &= 2. \\ \mathbf{x}_2 \cdot \mathbf{w} + b &= -1 \end{aligned}$$



But we know that  $\|\mathbf{x}_1 - \mathbf{x}_2\| = 2M$ .

Thus we have  $M = \frac{1}{\|\mathbf{w}\|}$ .

So, to maximize margin, minimize  $\|\mathbf{w}\|$ , subject to

$$y = \mathbf{x}_i \cdot \mathbf{w} + b \geq +1 \text{ for all positive instances } (y_i = +1)$$

$$y = \mathbf{x}_i \cdot \mathbf{w} + b \leq -1 \text{ for all negative instances } (y_i = -1)$$

Equivalently, minimize

$$\frac{1}{2} \|\mathbf{w}\|^2, \text{ subject to } t_i (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$$

## SVM

Find  $\mathbf{w}$  and  $b$  by doing the following minimization:

$$\min_{\mathbf{w}, b} \left( \frac{1}{2} \|\mathbf{w}\|^2 \right)$$

subject to :

$$t_i (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad i = 1, \dots, n$$

$$(t_i \in \{-1, +1\})$$

This is a quadratic optimization problem. Use “standard optimization tools” to solve it.

- **Dual formulation:** It turns out that  $\mathbf{w}$  can be expressed as a linear combination of a small subset of the training examples: those that lie exactly on margin:

$$\mathbf{w} = \sum_i y_i \alpha_i \mathbf{x}_i$$

such that  $\mathbf{x}_i$  lie exactly on the margin.

- These training examples are called “support vectors”. They carry all relevant information about the classification problem.

- The result of the SVM training algorithm (involving solving a quadratic programming problem), is the  $\alpha_i$ 's and the  $\mathbf{x}_i$ 's.

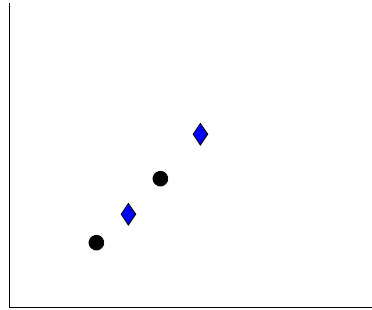
- For a new example  $\mathbf{x}$ , We can now classify  $\mathbf{x}$  using the support vectors:

$$\text{class}(\mathbf{x}) = \text{sgn} \left( \sum_i y_i \alpha_i \mathbf{x} \cdot \mathbf{x}_i + b \right)$$

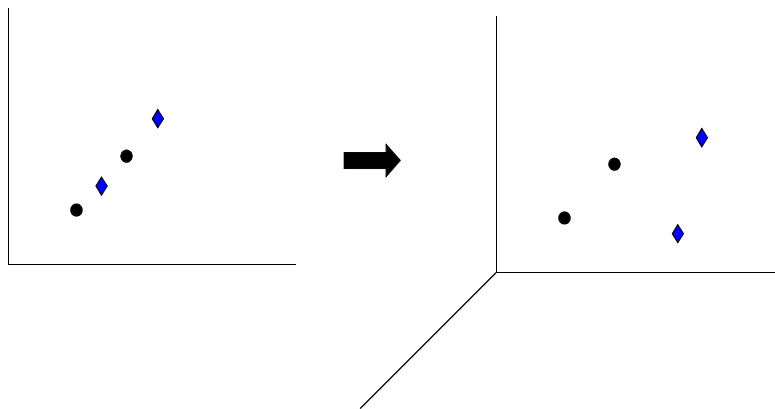
- This is the resulting SVM classifier.

## Non-linearly separable training examples

- What if the training examples are not linearly separable?



- Use old trick: find a function that maps points to a higher dimensional space (“feature space”) in which they are linearly separable, and do the classification in that higher-dimensional space.



Need to find a function  $\Phi$  that will perform such a mapping:

$$\Phi: \mathcal{X}^n \rightarrow F$$

Then can find hyperplane in higher dimensional feature space  $F$ , and do classification using that hyperplane in higher dimensional space.

- **Problem:**

- Recall that classification of instance  $\mathbf{x}$  is expressed in terms of dot products of  $\mathbf{x}$  and support vectors.

$$\text{Class}(\mathbf{x}) = \text{sgn} \left( \sum_i y_i \alpha_i (\mathbf{x} \cdot \mathbf{x}_i) + b \right)$$

- The quadratic programming problem of finding the support vectors and coefficients also depends only on dot products between training examples, rather than on the training examples outside of dot products.

- So if each  $\mathbf{x}_i$  is replaced by  $\Phi(\mathbf{x}_i)$  in these procedures, we will have to calculate a lot of dot products,  $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$

- But in general, if the feature space  $F$  is high dimensional,  $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$  will be expensive to compute.

- **Second trick:**

- Suppose that there were some magic function,

$$k(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

such that  $k$  is cheap to compute even though  $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$  is expensive to compute.

- Then we wouldn't need to compute the dot product directly; we'd just need to compute  $k$  during both the training and testing phases.
- The good news is: such  $k$  functions exist! They are called “kernel functions”, and come from the theory of integral operators.

Example: Polynomial kernel:

Suppose  $\mathbf{x} = (x_1, x_2)$  and  $\mathbf{y} = (y_1, y_2)$ .

$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^2$$

$$\text{Let } \Phi(\mathbf{x}) = (x_1^2, \sqrt{2} \cdot x_1 x_2, x_2^2).$$

Then :

$$\begin{aligned} \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}) &= \left( \begin{pmatrix} x_1^2 \\ \sqrt{2} \cdot x_1 x_2 \\ x_2^2 \end{pmatrix} \cdot \begin{pmatrix} y_1^2 \\ \sqrt{2} \cdot y_1 y_2 \\ y_2^2 \end{pmatrix} \right) \\ &= \left( \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \right)^2 = (\mathbf{x} \cdot \mathbf{y})^2 = k(\mathbf{x}, \mathbf{y}) \end{aligned}$$

## Recap of SVM algorithm

Given training set

$$S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m) \mid (\mathbf{x}_i, y_i) \in \mathfrak{X}^n \times \{+1, -1\}\}$$

1. Choose a map  $\Phi: \mathfrak{X}^n \rightarrow F$ , which maps  $\mathbf{x}_i$  to a higher dimensional feature space. (Solves problem that  $X$  might not be linearly separable in original space.)
2. Choose a cheap-to-compute kernel function
$$k(\mathbf{x}, \mathbf{z}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{z})$$
(Solves problem that in high dimensional spaces, dot products are very expensive to compute.)
3. Map all the  $\mathbf{x}_i$ 's to feature space  $F$  by computing  $\Phi(\mathbf{x}_i)$ .

4. Apply quadratic programming procedure (using the kernel function  $k$ ) to find a hyperplane  $(\mathbf{w}, w_0)$ , such that

$$\mathbf{w} = \sum_i y_i \alpha_i \Phi(\mathbf{x}_i)$$

where the  $\Phi(\mathbf{x}_i)$ 's are support vectors, the  $\alpha_i$ 's are coefficients, and  $w_0$  is a threshold, such that  $(\mathbf{w}, w_0)$  is the hyperplane maximizing the margin of  $S$  in  $F$ .

- Now, given a new instance,  $\mathbf{x}$ , find the classification of  $\mathbf{x}$  by computing

$$\begin{aligned} \text{class}(\mathbf{x}) &= \text{sgn} \left( \sum_i y_i \alpha_i (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}_i)) + w_0 \right) \\ &= \text{sgn} \left( \sum_i y_i \alpha_i k(\mathbf{x}, \mathbf{x}_i) + w_0 \right) \end{aligned}$$

### Most commonly used kernels

- Linear

$$K(\mathbf{x}, \mathbf{x}_i) = \mathbf{x} \cdot \mathbf{x}_i$$

- Polynomial

$$K(\mathbf{x}, \mathbf{x}_i) = [(\mathbf{x} \cdot \mathbf{x}_i) + 1]^d$$

- Gaussian (or “radial basis function”)

$$K(\mathbf{x}, \mathbf{x}_i) = e^{-\gamma |\mathbf{x} - \mathbf{x}_i|^2}$$

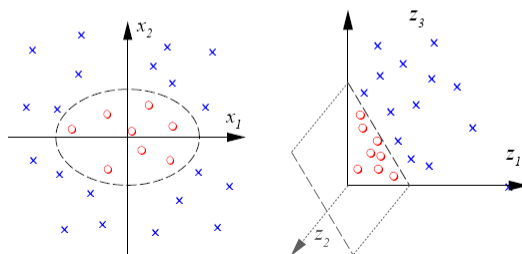
- Sigmoid

$$K(\mathbf{x}, \mathbf{x}_i) = \tanh(a\mathbf{x} \cdot \mathbf{x}_i + b)$$

## 21 SVMs : polynomial mapping

$$\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

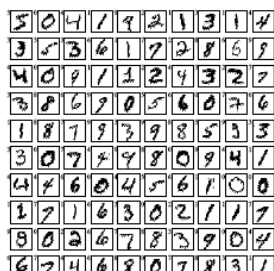


From [http://www1.cs.columbia.edu/~kathy/cs4701/documents/jason\\_svm\\_tutorial.pdf](http://www1.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf).

## 22 SVMs : non-linear case II

For example MNIST hand-writing recognition.  
60,000 training examples, 10000 test examples, 28x28.

Linear SVM has around 8.5% test error.  
Polynomial SVM has around 1% test error.

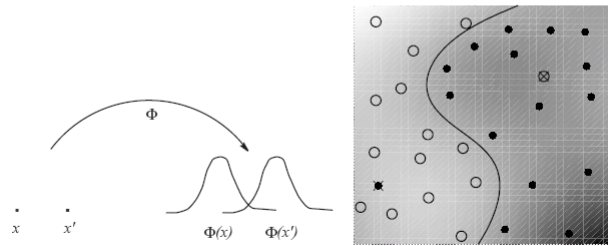


From [http://www1.cs.columbia.edu/~kathy/cs4701/documents/jason\\_svm\\_tutorial.pdf](http://www1.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf).

## 28 RBF-SVMs

The RBF kernel  $K(x, x') = \exp(-\gamma\|x - x'\|^2)$  is one of the most popular kernel functions. It adds a "bump" around each data point:

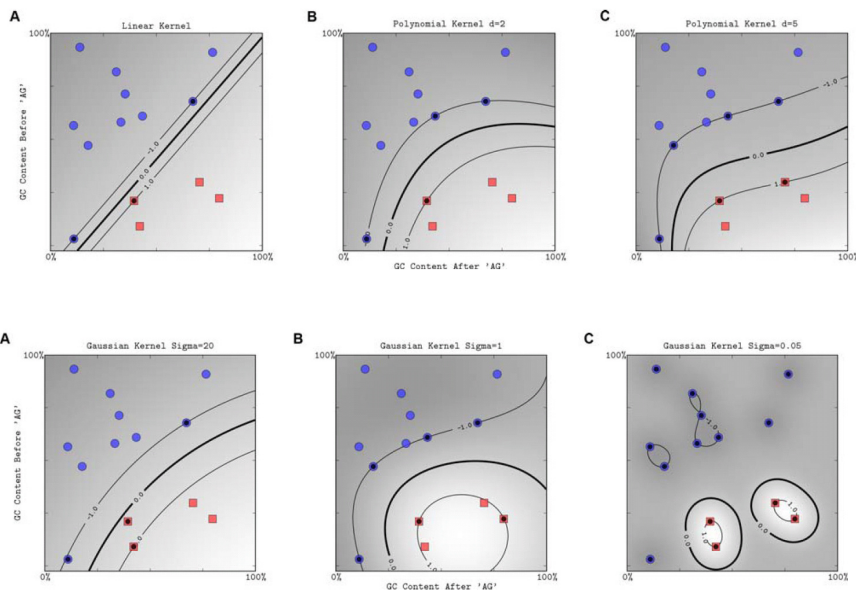
$$f(x) = \sum_{i=1}^m \alpha_i \exp(-\gamma\|x_i - x\|^2) + b$$



Using this one can get state-of-the-art results.

From [http://www1.cs.columbia.edu/~kathy/cs4701/documents/jason\\_svm\\_tutorial.pdf](http://www1.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf).

## Linear, polynomial, and Gaussian kernels



Demo: SVM\_Light