Sketched Derivation of error bound using VC-dimension (1)

Bound our usual PAC expression by the probability that an algorithm has 0 error on the training examples S but high error on a second random sample $\hat{S}$, using a Chernoff bound:

$$\Pr\left[(\exists h \in H)\left(error_{\mathcal{D}}(h) > \varepsilon \wedge error_S(h) = 0\right)\right] \leq$$
$$2\Pr\left[(\exists h \in H)\left(error_{\hat{S}}(h) > \varepsilon m/2 \wedge error_S(h) = 0\right)\right]$$

Then given a fixed sample of $2m$ examples, the fraction of permutations with all errors in the second half is at most:

$$\frac{1}{2^{\varepsilon m/2}}$$

(derivation based on Cristianini and Shawe-Taylor, 1999)

Sketched Derivation of error bound using VC-dimension (2)

Now, we just need a bound on the size of the hypothesis space when restricted to 2*m* examples. Define a growth function:

$$growth_H(m) = \max_{(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m)} \left| \left\{ h(\mathbf{x}_1), h(\mathbf{x}_2), \ldots, h(\mathbf{x}_m) : h \in H \right\} \right|$$

So if sets of all sizes can be shattered by *H*, then $growth_H(m) = 2^m$

Last piece of VC theory is a bound on the growth function in terms of VC dimension *d*:

$$growth_H(m) \le \left(\frac{em}{d} \right)^d$$

Sketched Derivation of error bound using VC-dimension (3)

Putting these pieces together:

$$\Pr\left[(\exists h \in H)\left(error_{\mathcal{D}}(h) > \varepsilon \wedge error_S(h) = 0\right)\right] \leq$$

$$2\left(\frac{2em}{d}\right)^{d} 2^{-\varepsilon m/2}$$

In terms of a bound on error:

$$error_{\mathcal{D}}(h) \leq \frac{2}{m}\left(d \log \frac{2em}{d} + \log \frac{2}{\delta}\right)$$

Support Vector Machines

The Learning Problem

- Set of *m* training examples: $(\mathbf{x}_i, y_i)$
- Where $\mathbf{x}_i \in \mathbf{R}^n \quad y_i \in \{-1, 1\}$

SVMs are perceptrons that work in a derived feature space and maximize margin.

Perceptrons

A linear learning machine, characterized by a vector of real-valued weights **w** and bias *b*:

$$f(\mathbf{x}) = \mathrm{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$

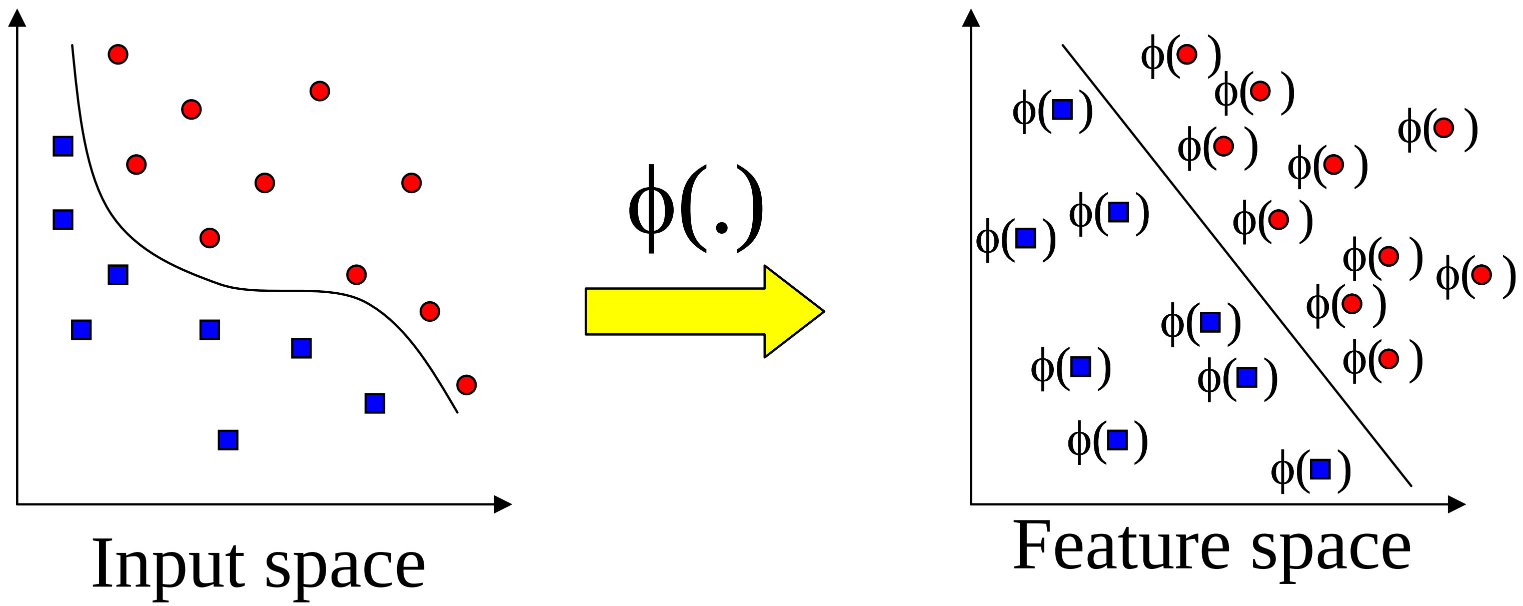
Learning algorithm – repeat until no mistakes are made:

```
for $i = 1$ to $m$
    if $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \le 0$
        $\mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i$
        $b \leftarrow b + \eta y_i$
```

Derived Features

Linear Perceptrons can’t represent XOR.

Solution –map to a derived feature space:



(from http://www.cse.msu.edu/~lawhiu/intro_SVM.ppt)

Derived Features

With the derived feature $x_1 x_2$, XOR becomes linearly separable!

…maybe for another problem, we need $x_1{}^7 x_2{}^{12}$

Large feature spaces =>

1) Inefficiency
2) Overfitting

Perceptrons (dual form)

- **w** is a linear combination of training examples, and
- Only really need **dot products** of feature vectors

Standard form:

```
for $i = 1$ to $m$
    if $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \leq 0$
        $\mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i$
        $b \leftarrow b + \eta y_i$
```

Dual form:

```
for $i = 1$ to $m$
    if $y_i\left(\sum_{j=1}^{m} \alpha_j y_j (\mathbf{x}_j \cdot \mathbf{x}_i) + b\right) < 0$
        $\alpha_i \leftarrow \alpha_i + \eta$
        $b \leftarrow b + \eta y_i$
```

Kernels (1)

In the dual formulation, features only enter the computation in terms of dot products:

$$f(\mathbf{x}_i) = \left(\sum_{j=1}^{l} \alpha_j y_j (\mathbf{x}_j \cdot \mathbf{x}_i) + b \right)$$

In a derived feature space, this becomes:

$$f(\mathbf{x}_i) = \left(\sum_{j=1}^{l} \alpha_j y_j (\phi(\mathbf{x}_j) \cdot \phi(\mathbf{x}_i)) + b \right)$$

Kernels (2)

The kernel trick – find an easily-computed function K such that:

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

K makes learning in feature space efficient:

$$f(\mathbf{x}_i) = \left(\sum_{j=1}^{m} \alpha_j y_j K(\mathbf{x}_j, \mathbf{x}_i) + b \right)$$

We avoid explicitly evaluating $\phi(\mathbf{x})$!

Kernel Example

Let $K(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}')^2$

$$\begin{aligned}
K([x_1, x_2] \cdot [x_1', x_2']) &= ([x_1, x_2] \cdot [x_1', x_2'])^2 \\
&= (x_1 x_1' + x_2 x_2')^2 \\
&= x_1 x_1' x_1 x_1' + 2 x_1 x_1' x_2 x_2' + x_2 x_2' x_2 x_2' \\
&= x_1{}^2 x_1'^2 + 2 x_1 x_2 x_1' x_2' + x_2{}^2 x_2'{}^2 \\
&= \left(\left[x_1{}^2, \sqrt{2} x_1 x_2, x_2{}^2\right] \cdot \left[x_1'^2, \sqrt{2} x_1' x_2', x_2'{}^2\right]\right) \\
&= \phi([x_1, x_2]) \cdot \phi([x_1', x_2'])
\end{aligned}$$

Where: $\phi([x_1, x_2]) = \left[x_1{}^2, \sqrt{2} x_1 x_2, x_2{}^2\right]$

…(we can do XOR!)

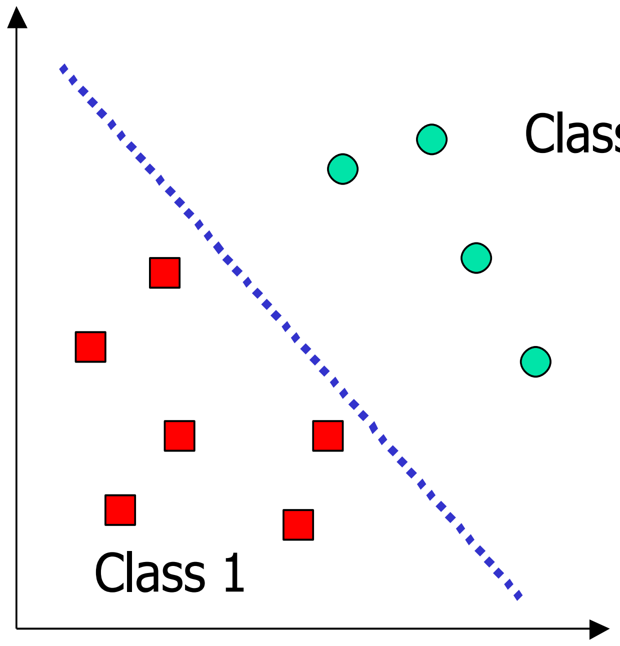
Kernel Examples

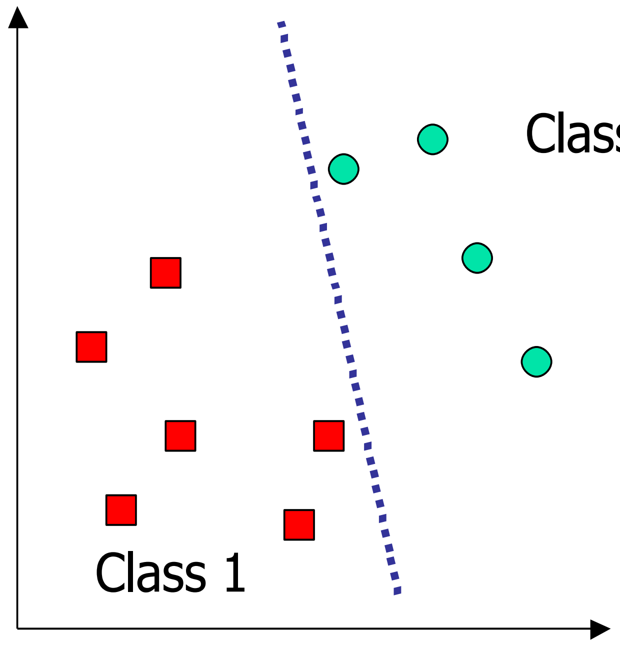
$$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + 1)^d$$ -- Polynomial Kernel (hypothesis space is all polynomials up to degree *d*). VC dimension gets large with *d*.

$$K(\mathbf{x}, \mathbf{x}') = e^{-\|\mathbf{x}-\mathbf{x}'\|^2 / \sigma^2}$$ -- Gaussian Kernel (hypotheses are 'radial basis function networks'). VC dimension is infinite.

With such high VC dimension, how can SVMs avoid overfitting?

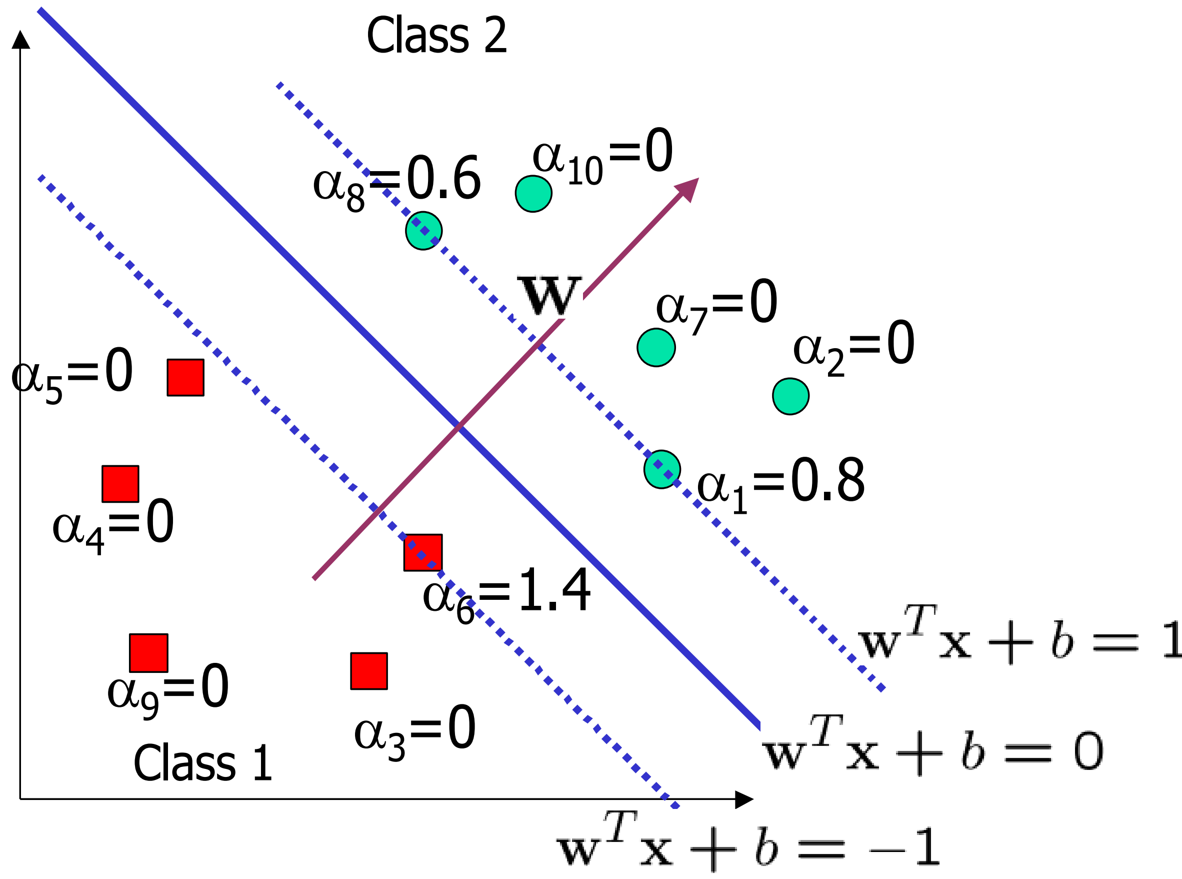
‘Bad’ separators





Margin

Margin – minimum distance between the separator and an example. Hence, only some examples (the 'support vectors') actually matter.

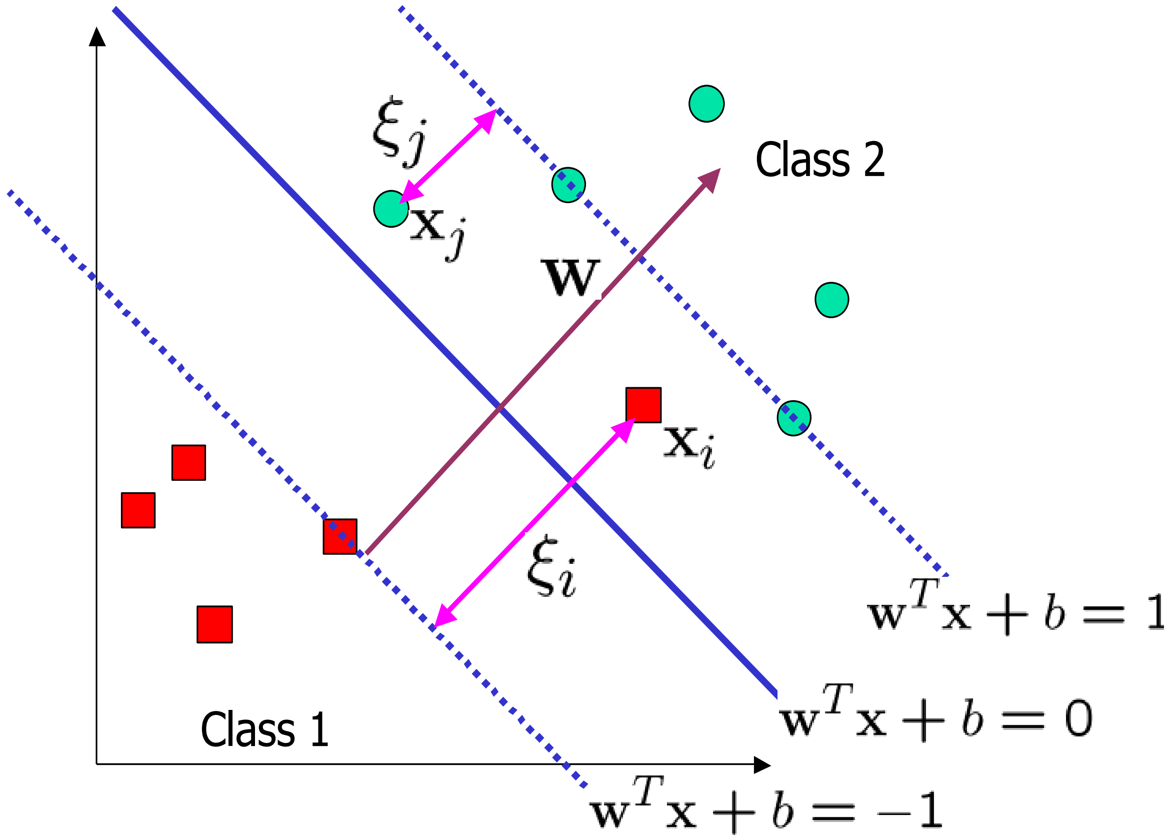


(from http://www.cse.msu.edu/~lawhiu/intro_SVM.ppt)

Slack Variables

What if data is not separable?

Slack variables – allow training points to move normal to separating hyperplane with some penalty.



ξ_j
$\mathbf{x}_j$
Class 2
$\mathbf{w}$
$\mathbf{x}_i$
ξ_i
$\mathbf{w}^T\mathbf{x}+b=1$
$\mathbf{w}^T\mathbf{x}+b=0$
Class 1
$\mathbf{w}^T\mathbf{x}+b=-1$

(from http://www.cse.msu.edu/~lawhiu/intro_SVM.ppt)

Avoiding Overfitting

PAC bounds can be found in terms of margin (instead of VC dimension).

Thus, SVMs find the separating hyperplane of maximum margin.

(Burges, 1998) gives an example in which performance improves for Gaussian kernals when σ is chosen according to a generalization bound.

Finding the maximum margin hyperplane

Minimize $\|\mathbf{w}\|^2 + C\left(\sum_i \xi_i\right)$

Subject to the constraints that

$$\mathbf{x}_i \cdot \mathbf{w} + b \geq +1 - \xi_i \quad \text{for} \quad y_i = +1$$

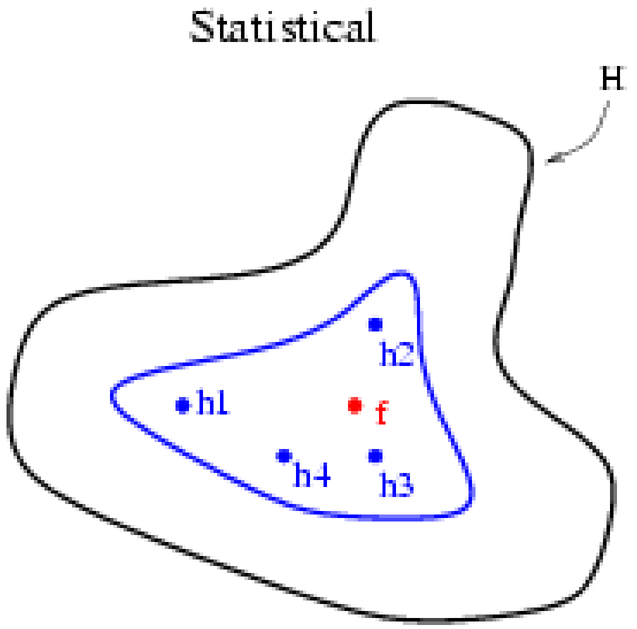
$$\mathbf{x}_i \cdot \mathbf{w} + b \leq -1 + \xi_i \quad \text{for} \quad y_i = -1$$

$$\xi_i \geq 0$$

This can be expressed as a convex quadratic program.

Flashback to Boosting

- One justification for boosting –averaging over several hypotheses *h* helps to find the true concept *f*.
- Similar to *f* having maximum margin – indeed, boosting does maximize margin.



From (Dietterich, 2000)

Fine Print

$$\text{Minimize } \|\mathbf{w}\|^2 + C\left(\sum_i \xi_i\right)$$

- We have to choose C
- We also have to choose our kernel and its parameters (e.g. Gaussian width)
- Use cross validation!

References

Martin Law's tutorial, *An Introduction to Support Vector Machines:* http://www.cse.msu.edu/~lawhiu/intro_SVM.ppt

(Christianini and Taylor, 1999) Nello Cristianini , John Shawe-Taylor, *An introduction to support Vector Machines: and other kernel-based learning methods*, Cambridge University Press, New York, NY, 1999

(Burges, 1998) C. J. C. Burges, *A tutorial on support vector machines for pattern recognition*," Data Mining and Knowledge Discovery, vol. 2, no. 2, pp. 1-47, 1998

(Dietterich, 2000) Thomas G. Dietterich, Ensemble Methods in Machine Learning, Proceedings of the First International Workshop on Multiple Classifier Systems, p.1-15, June 21-23, 2000