

## Introduction to Computational Learning Theory (COLT)

### **PAC (Probably Approximately Correct) Learning**

Questions addressed:

- What is “successful learning”?
- How many training examples are needed for successful learning?
- How much computation time is needed for successful learning?

### **Model Complexity and Vapnik-Chervonenkis (VC) Dimension**

(work largely due to Vladimir Vapnik, collected in his  
book

*The Nature of Statistical Learning Theory, 1995*

- **Occam’s razor:** Given two hypotheses (models) that correctly classify the training data, we should select the less complex one.
- But how do we measure model complexity in a comparable way for any hypothesis?

- **A general framework:** Let  $X$  be a space of instances, and let  $H$  be a space of hypothesis (e.g., all linear decision surfaces in two dimensions).
- We will measure the complexity of  $h \in H$  using **Vapnik-Chervonenkis (VC) Dimension**.

### Vapnik-Chervonenkis (VC) Dimension

- Denoted  $VC(H)$
- Measures the largest subset  $S$  of distinct instances  $\mathbf{x} \in X$  that can be represented using  $H$ .
- “Represented using  $H$ ” means:  
For each dichotomy of  $S$ ,  $\exists h \in H$  such that  $h$  classifies  $\mathbf{x} \in S$  perfectly according to this dichotomy.

## Dichotomies

- Let  $S$  be a subset of instances,  $S \subseteq X$ .
- Each  $h \in H$  partitions  $S$  into two subsets:

$$\{\mathbf{x} \in S \mid h(\mathbf{x}) = 1\}$$

$$\{\mathbf{x} \in S \mid h(\mathbf{x}) = 0\}$$

- This partition is called a “**dichotomy**”.

## Example

- Let  $X = \{(0,0), (0, 1), (1,0), (1,1)\}$ .
- Let  $S = X$
- What are the dichotomies of  $S$  ?
- In general, given  $S \subseteq X$ , how many possible dichotomies are there of  $S$  ?

## Shattering a Set of Instances

- Definition :
  - $H$  **shatters**  $S$  if and only if  $H$  can represent all dichotomies of  $S$ .
  - That is,  $H$  **shatters**  $S$  if and only if for all dichotomies of  $S$ ,  $\exists h \in H$  that is consistent with that dichotomy.
- **Example:** Let  $S$  be as on the previous slide, and let  $H$  be the set of all linear decision surfaces in two dimensions. Does  $H$  shatter  $S$  ?
- Let  $H$  be the set of all multilayer perceptrons. Does  $H$  shatter  $S$ ?

- Comments on the notion of **shattering**:
  - Ability of  $H$  to shatter a set of instances  $S$  is measure of  $H$ 's capacity to represent target concepts defined over these instances.
  - **Intuition:** The larger the subset of  $X$  that can be shattered, the more expressive  $H$  is. This is what VC dimension measures.

## VC Dimension

Definition:

$\text{VC}(H)$ , defined over  $X$ , is the size of the largest finite subset  $S \subseteq X$  shattered by  $H$ .

## Examples of VC Dimension

1. Let  $X = \mathfrak{R}$  (e.g.,  $x = \text{height of some person}$ ), and  $H = \text{set of intervals on the real number line}$ :

$$H = \{a < x < b \mid a, b \in \mathfrak{R}\}.$$

In other words, for each  $h \in H$ ,  $h = (a, b)$ , for some real-valued  $a, b$ .

$$h(x) = \begin{cases} 1 & \text{if } a < x < b \\ 0 & \text{otherwise} \end{cases}$$

What is  $\text{VC}(H)$ ?

- Need to find largest subset of  $X$  that can be shattered by  $H$ .

- Consider subset of  $X$  containing two instances:

$$S = \{2.2, 4.5\}$$

Can  $S$  be shattered by  $H$ ?

Yes. Write down different dichotomies of  $S$  and the hypotheses that are consistent with them.

- This shows  $VC(H) \geq 2$ .
- Is there a subset,  $S = \{x_0, x_1, x_2\}$ , of size 3 that can be shattered? Show that the answer is “no”.
- Thus  $VC(H) = 2$ .

2. Let  $X = \{(x, y) \mid x, y \in \mathfrak{R}\}$ . Let  $H =$  set of all linear decision surfaces in the plane (i.e., lines that separate examples into positive and negative classifications).

What is  $VC(H)$ ?

- Show that there is a set of two points that can be shattered by constructing all dichotomies and showing that there is a  $h \in H$  that represents each one.
- Is there a set of three points that can be shattered?  
Yes, if not co-linear.
- Is there a set of four points that can be shattered?  
Why or why not?
- More generally: the VC dimension of linear decision surfaces in an  $r$ -dimensional space (i.e., the VC dimension of a perceptron with  $r$  inputs) is  $r + 1$ .

## Finding VC dimension

General approach:

- To show that  $VC(H) \geq m$ , show that there exists a subset of  $m$  instances that is shattered by  $H$ .
- Then, to show that  $VC(H) = m$ , show that no subset of  $m+1$  instances is shattered by  $H$ .

## Can we use VC dimension to design better learning algorithms?

- Vapnik proved: Given a target function  $t$  and a sample  $S$  of  $m$  training examples drawn independently using  $D$ , for any  $\eta$  between 0 and 1, with probability  $1-\eta$ :

$$\text{error}_D(h) \leq \text{error}_S(h) + \sqrt{\frac{\text{VC}(H)(\log_2(2m/\text{VC}(H))+1) - \log_2(\eta/4)}{m}}$$

- This gives an upper bound on true error as a function of training error, number of training examples, VC dimension of  $H$ , and  $\eta$ .
- Thus, for a given  $\eta$ , we can compute tradeoff among expected true error, number of training examples needed, and VC dimension.

- Need  $\text{VC}(H)$  high enough so that  $H$  can represent target concept (e.g., quadratic polynomials instead of lines), but not so high that the learner will overfit (e.g., degree-100 polynomials).

## Structural Risk Minimization Principle

- Principle: A function that fits the training data and minimizes VC dimension will generalize well.
- In other words, to minimize true error, both training error and the VC-dimension term should be small
- Large model complexity can lead to overfitting and high generalization error.

“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.)

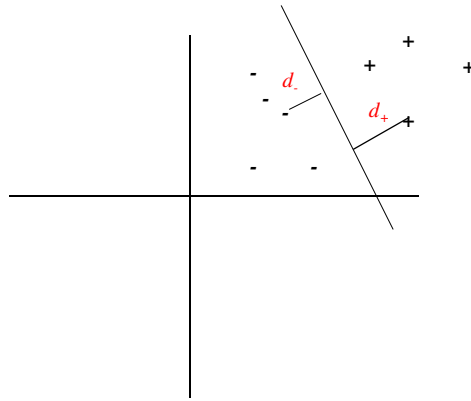
That “shining peak” is the 1979 paper by V. Vapnik, “Estimation of dependences based on empirical data” (in Russian).

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.

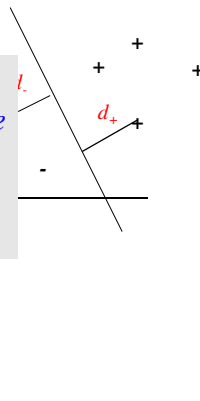
Back to support vector machines

- The margin of the training set  $S$  with respect to the hyperplane is  $d_+ + d_-$ .



- The margin of the training set  $S$  with respect to the hyperplane is  $d_+ + d_-$ .

*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.*



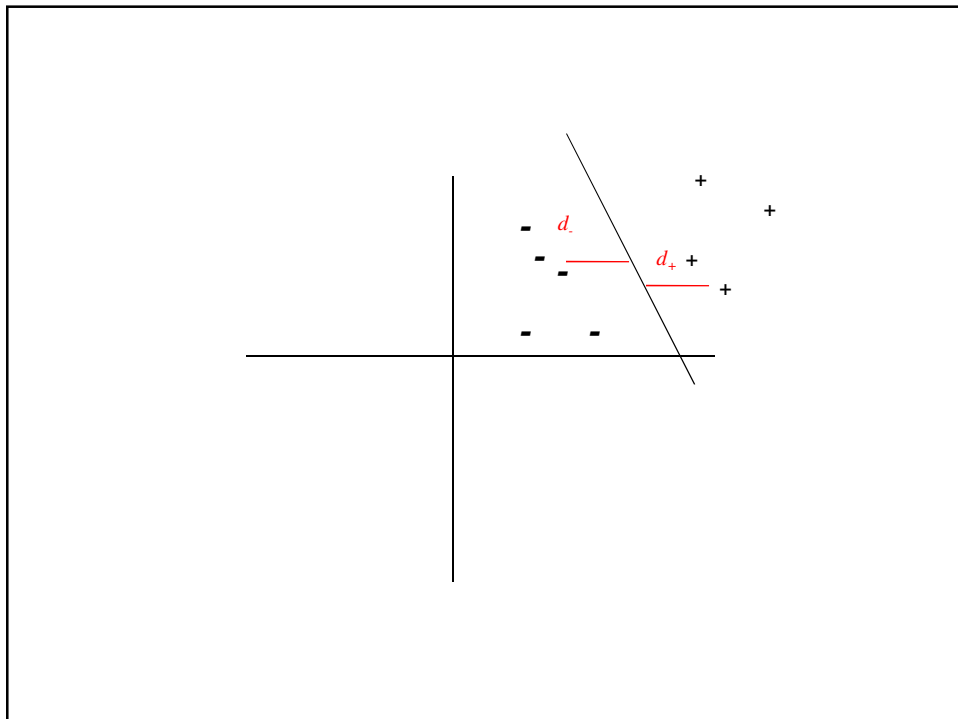
- Note that the hyperplane is defined as

$$\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$$

- To make the math easier, we will adjust  $b$  such that the hyperplane is halfway in between the closest positive and negative examples, and

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

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



- In this case,

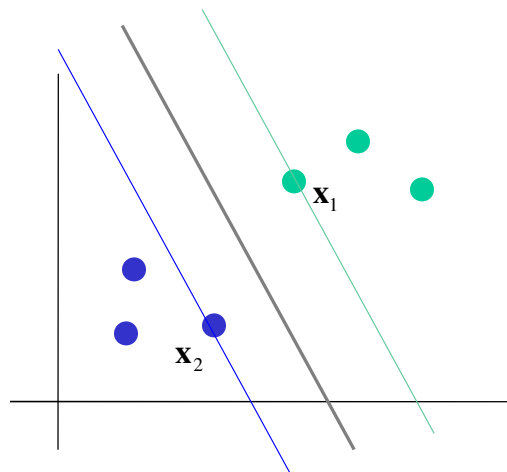


$$d_+ = d_- = \frac{1}{\|\mathbf{w}\|} \equiv \frac{1}{\sqrt{\mathbf{w} \cdot \mathbf{w}}}$$

- In this case,



$$d_+ = d_- = \frac{1}{\|\mathbf{w}\|} \equiv \frac{1}{\sqrt{\mathbf{w} \cdot \mathbf{w}}}$$



- In this case,

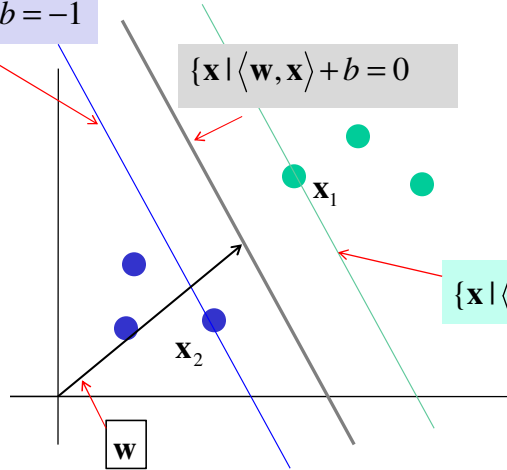


$$d_+ = d_- = \frac{1}{\|\mathbf{w}\|} \equiv \frac{1}{\sqrt{\mathbf{w} \cdot \mathbf{w}}}$$

$$\{\mathbf{x} \mid \langle \mathbf{w}, \mathbf{x} \rangle + b = -1\}$$

$$\{\mathbf{x} \mid \langle \mathbf{w}, \mathbf{x} \rangle + b = 0\}$$

$$\{\mathbf{x} \mid \langle \mathbf{w}, \mathbf{x} \rangle + b = +1\}$$



- In this case,



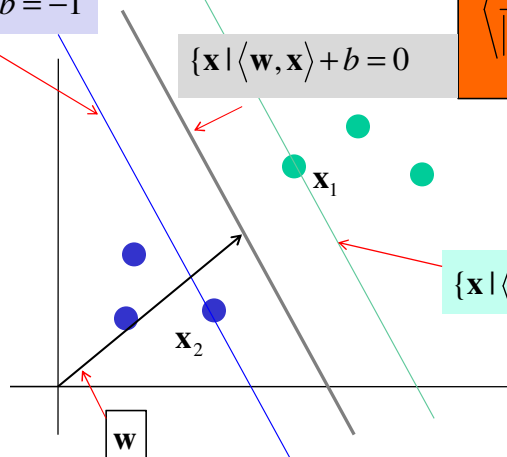
$$d_+ = d_- = \frac{1}{\|\mathbf{w}\|} \equiv \frac{1}{\sqrt{\mathbf{w} \cdot \mathbf{w}}}$$

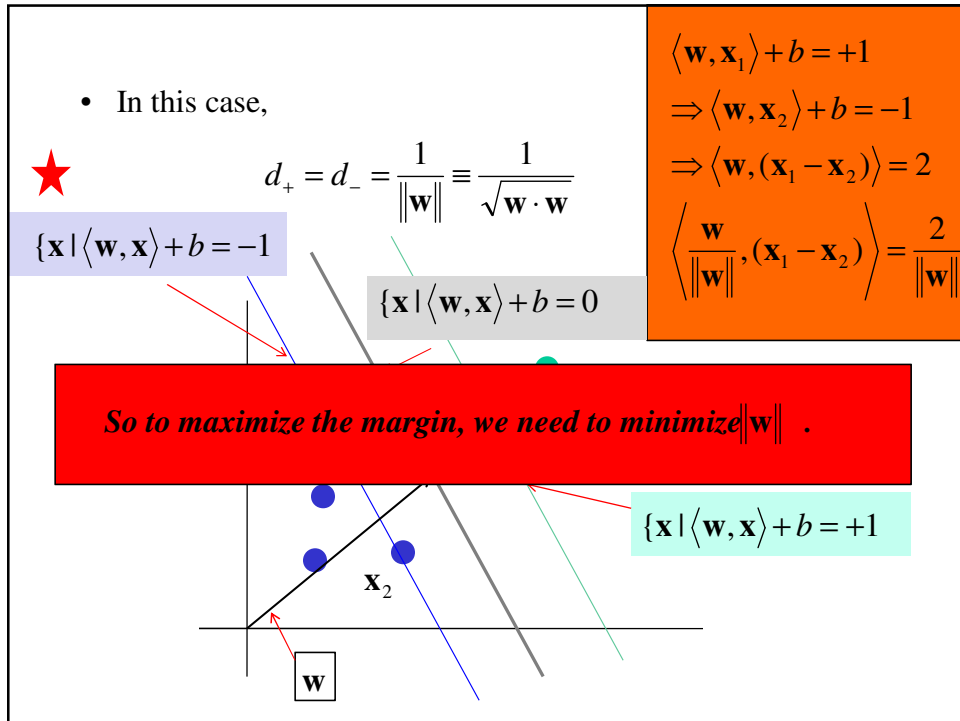
$$\{\mathbf{x} \mid \langle \mathbf{w}, \mathbf{x} \rangle + b = -1\}$$

$$\{\mathbf{x} \mid \langle \mathbf{w}, \mathbf{x} \rangle + b = 0\}$$

$$\{\mathbf{x} \mid \langle \mathbf{w}, \mathbf{x} \rangle + b = +1\}$$

$$\begin{aligned} \langle \mathbf{w}, \mathbf{x}_1 \rangle + b &= +1 \\ \Rightarrow \langle \mathbf{w}, \mathbf{x}_2 \rangle + b &= -1 \\ \Rightarrow \langle \mathbf{w}, (\mathbf{x}_1 - \mathbf{x}_2) \rangle &= 2 \\ \left\langle \frac{\mathbf{w}}{\|\mathbf{w}\|}, (\mathbf{x}_1 - \mathbf{x}_2) \right\rangle &= \frac{2}{\|\mathbf{w}\|} \end{aligned}$$





## Hard-Margin 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:

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

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

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

## Soft-Margin SVM

- Use “slack variables” to allow some errors on the training set. See description in assigned reading.

- **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 (minimum distance to hyperplane):

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

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:

$$\text{class}(\mathbf{x}) = \text{sgn}\left(\sum_i \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle + b\right)$$

- This is the resulting SVM classifier.

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

$$\text{class}(\mathbf{x}) = \text{sgn}\left(\sum_i \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle + b\right)$$

- This is the resulting SVM classifier.

- **Problem:**

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

$$\text{Class}(\mathbf{x}) = \text{sgn}\left(\sum_i \alpha_i (\mathbf{x} \cdot \mathbf{x}_i) + w_0\right)$$

- 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.
- Also  $\Phi(\mathbf{x})$  can 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  $\mathbf{y} = (y_1, y_2)$ .

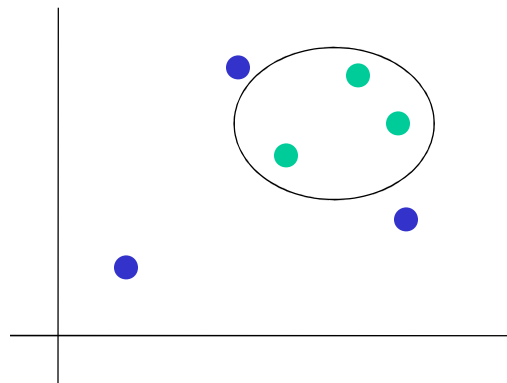
$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^2$$

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

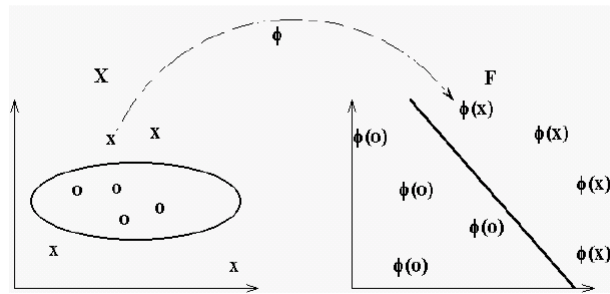
Then :

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

**Possible decision boundary using polynomial kernel**



## Example: Polynomial Kernels



[www.support-vector.net](http://www.support-vector.net)

## 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

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

## 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

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

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

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

## Measures of Performance

- Accuracy (percent correct over all test instances)
- Precision/Recall
- Area under ROC curve

## Precision / Recall

- Confusion matrix for a classifier:

	<b>Classified Positive</b>	<b>Classified Negative</b>
<b>Positive Examples</b>	True positives (TP)	False negatives (FN)
<b>Negative Examples</b>	False positives (FP)	True negatives (TN)

## Definition of performance measures

- **Accuracy:** proportion of classifications, over all the  $N$  examples, that were correct:

$$\text{accuracy} = \frac{TP + TN}{N}$$

- **Recall (or true positive rate, or “detection rate”):** Proportion of positive examples that were classified correctly:

$$\text{recall} = \frac{TP}{TP + FN}$$

- **False positive rate:** Proportion of negative examples that were incorrectly classified as positive:

$$\text{false positive rate} = \frac{FP}{TN + FP}$$

- **True negative rate:** Proportion of negative examples that were classified correctly:

$$\text{true negative rate} = \frac{TN}{TN + FP}$$

- **False negative rate:** Proportion of positive examples that were incorrectly classified as negative:

$$\text{false negative rate} = \frac{FN}{TP + FN}$$

- **Precision (or “sensitivity”):** Proportion of correct positive classifications over all positive classifications:

$$\text{precision} = \frac{TP}{TP + FP}$$

## Interpretation of precision and recall

- Precision and recall are often plotted against one another, especially in “detection” applications (such as spam detection), when positive examples are sparse in the observed data.
- **Recall (or detection rate):** How often did the system correctly identify positive examples when it encountered them?
- **Precision (or sensitivity):** How often did the system get its own positive classifications correct?
- How do these two measures trade off against one another?

### Precision / Recall Curves

