

[80245013 Machine Learning, Fall, 2019]

# **Supervised Learning Classification**

**Jun Zhu**

`dcszj@mail.tsinghua.edu.cn`

`http://ml.cs.tsinghua.edu.cn/~jun`

Institute for AI

Tsinghua University

September 17, 2019

# Supervised Learning

◆ **Task:** learn a predictive function

$$h : \mathcal{X} \rightarrow \mathcal{Y}$$

**Feature** space  $\mathcal{X}$

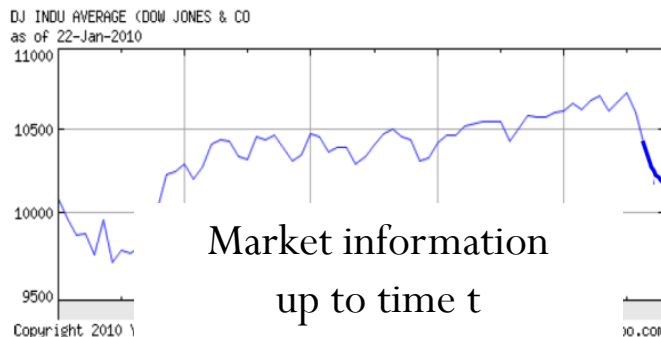
**Label** space  $\mathcal{Y}$



Words in documents



“Sports”  
“News”  
“Politics”  
...



Share price  
“\$ 20.50”

◆ “Experience” or training data:

$$\{ \langle \mathbf{x}_d, y_d \rangle \}_{d=1}^D, \quad \mathbf{x}_d \in \mathcal{X}, y_d \in \mathcal{Y}$$

# Supervised Learning – classification

Feature space  $\mathcal{X}$

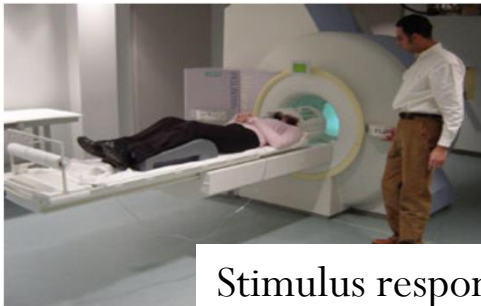
Label space  $\mathcal{Y}$



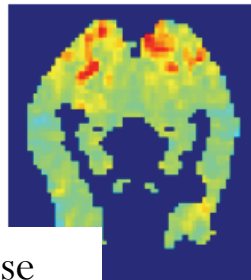
Words in documents



“Sports”  
“News”  
“Politics”  
...



Stimulus response



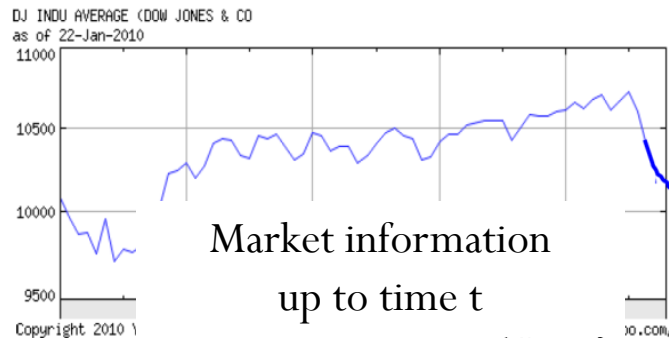
“Tool”  
“Animal”  
...

Discrete Labels

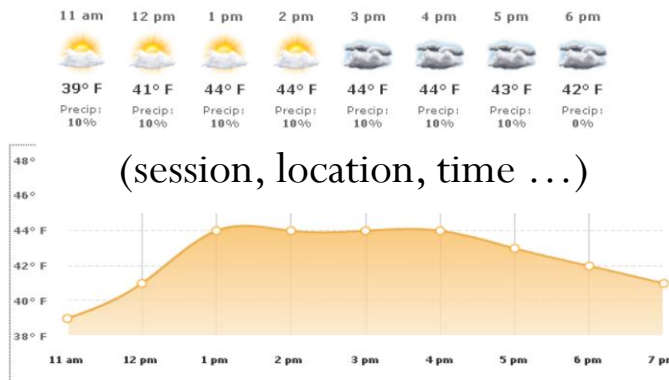
# Supervised Learning – regression

Feature space  $\mathcal{X}$

Label space  $\mathcal{Y}$



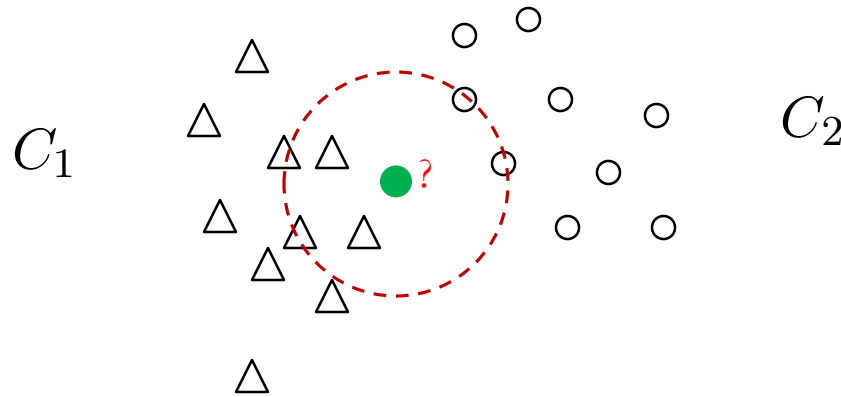
Share price  
“\$ 20.50”



Temperature  
“42° F”

Continuous Labels

# How to learn a classifier?



***K-NN***: a Non-parametric approach

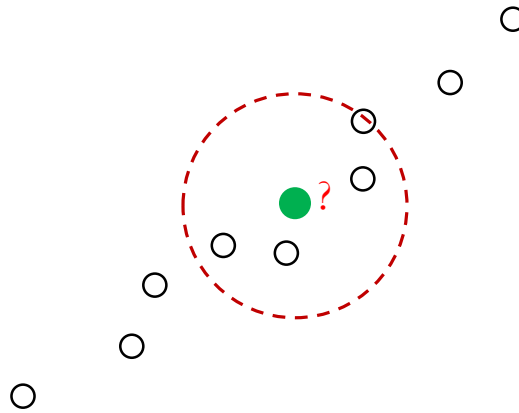
# Properties of K-NN

- ◆ Simple
- ◆ Strong consistency results:
  - With infinite data, the error rate of K-NN is at most twice the optimal error rate (i.e., Bayes error rate)
- ◆ **Note:** Bayes error rate — the minimum achievable error rate given the distribution of the data

# Issues of K-NN

- ◆ Computationally intensive for large training sets
  - Clever nearest neighbor search helps
- ◆ Selection of K
- ◆ Distance metric matters a lot
  - Aware of the metric learning field

# K-NN for regression



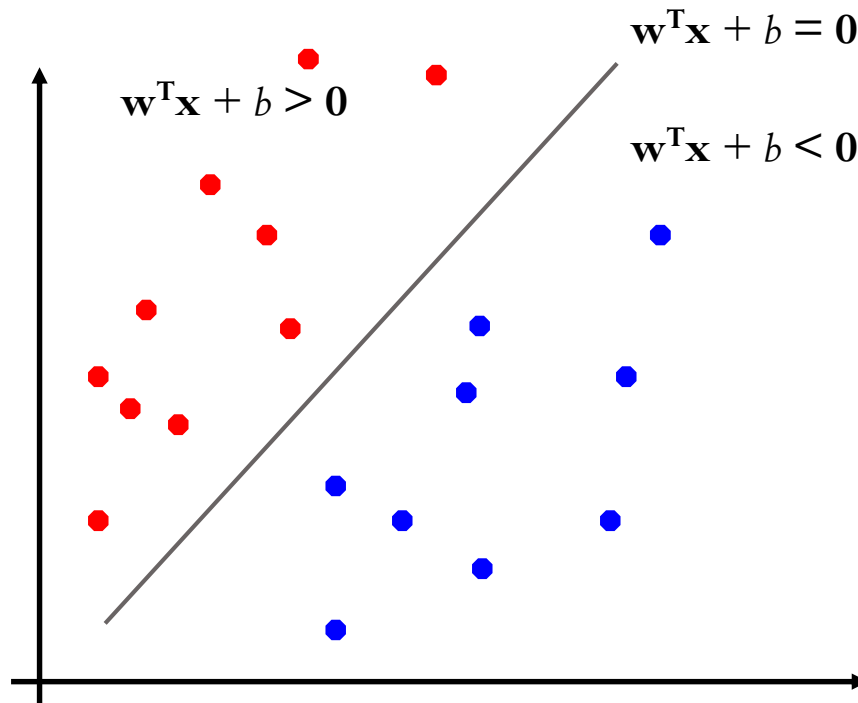
$$\hat{y} = \sum_{i \in \mathcal{N}_K} \frac{1}{\text{dist}(\mathbf{x}, \mathbf{x}_i)} y_i$$

*A weighted average is an estimate; where the weight is the inverse distance*



# A Parametric Method

- Binary classification can be viewed as the task of separating classes in feature space:



$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x} + b)$$

# A brief history

## ◆ Fisher's linear discriminator (1936)

- Log-ratio of two Gaussians

$$F_{sq}(\mathbf{x}) = \text{sign} \left[ \frac{1}{2}(\mathbf{x} - \mathbf{m}_1)^T \Sigma_1^{-1}(\mathbf{x} - \mathbf{m}_1) - \frac{1}{2}(\mathbf{x} - \mathbf{m}_2)^T \Sigma_2^{-1}(\mathbf{x} - \mathbf{m}_2) + \ln \frac{|\Sigma_2|}{|\Sigma_1|} \right]$$

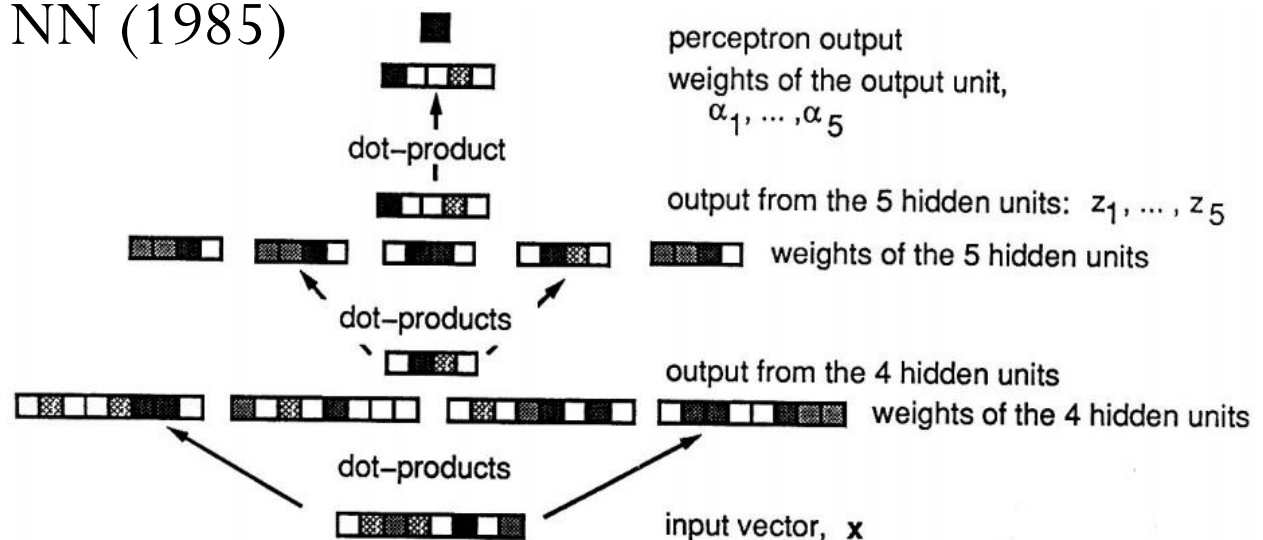
## ◆ Rosenblatt's perceptron (1962)

- Only top-layer is updated

## ◆ Back-prop for NN (1985)

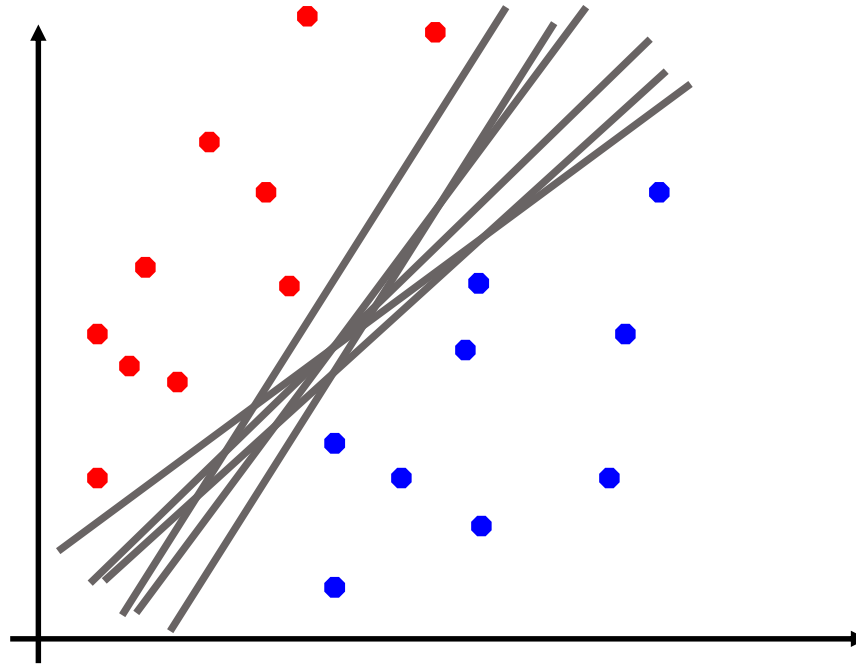
## ◆ SVM (1995)

## ◆ ...



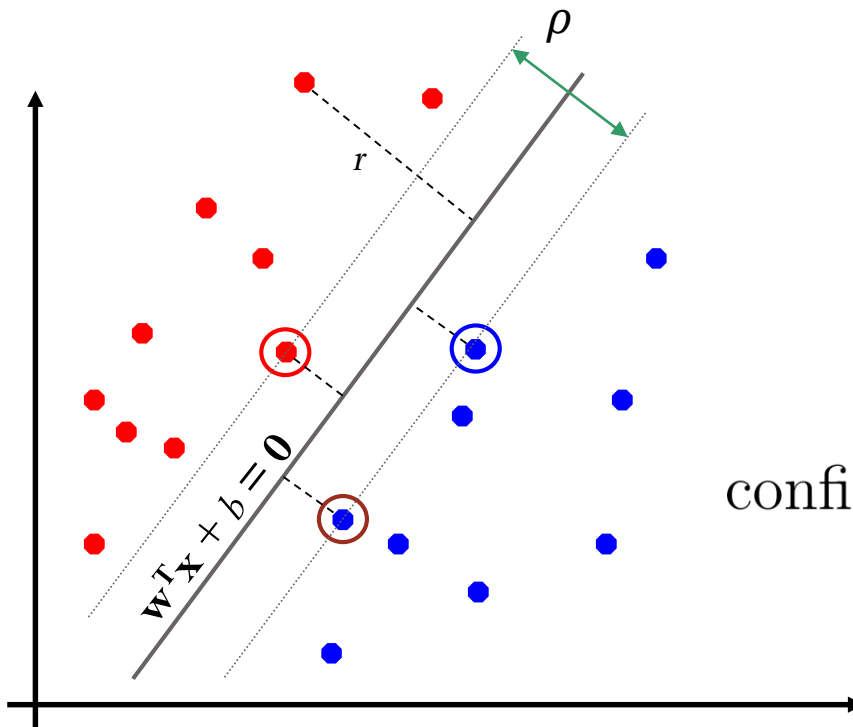
# Linear Separators

◆ Which of the linear separators is optimal?



# Classification Margin

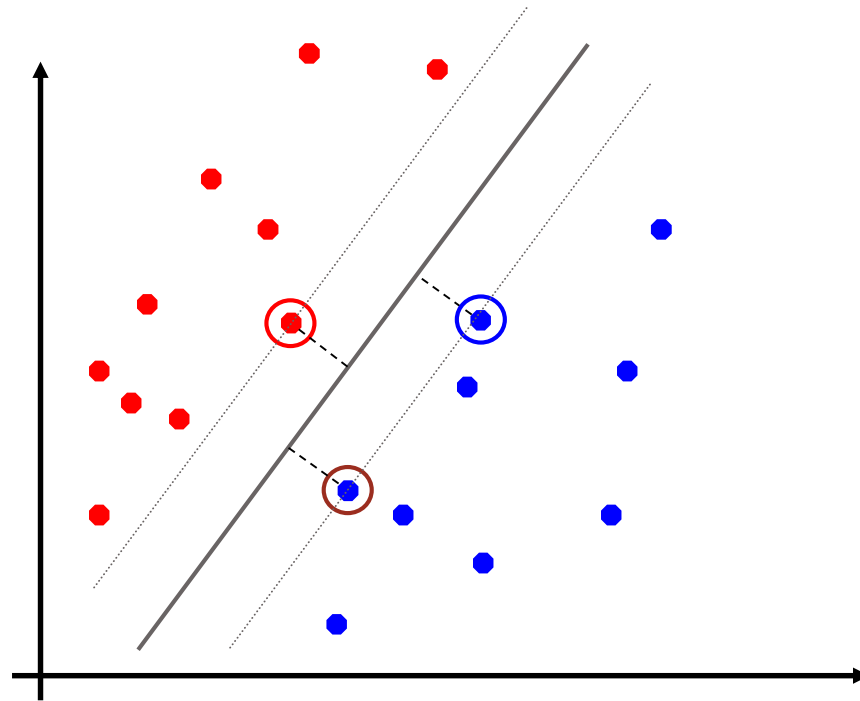
- Distance from example  $\mathbf{x}_i$  to the separator is  $r = \frac{|\mathbf{w}^\top \mathbf{x}_i + b|}{\|\mathbf{w}\|}$
- Examples closest to the hyperplane are *support vectors*.
- Margin  $\rho$*  of the separator is the distance between supporting hyperplanes.



$$\text{confidence} := y_i(\mathbf{w}^\top \mathbf{x}_i + b)$$

# Max-margin Classification

- ◆ Maximizing the margin is good according to intuition and PAC theory.
- ◆ Implies that only support vectors matter; other training examples are ignorable.



# Linear SVM

- ◆ Let training set  $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ ,  $\mathbf{x}_i \in \mathbb{R}^d$ ,  $y_i \in \{-1, +1\}$  be separated by a hyperplane with margin  $\rho$ . Then for each training example  $(\mathbf{x}_i, y_i)$ :

$$\begin{aligned} \mathbf{w}^T \mathbf{x}_i + b &\leq -\rho/2 & \text{if } y_i = -1 \\ \mathbf{w}^T \mathbf{x}_i + b &\geq \rho/2 & \text{if } y_i = 1 \end{aligned} \iff y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq \rho/2$$

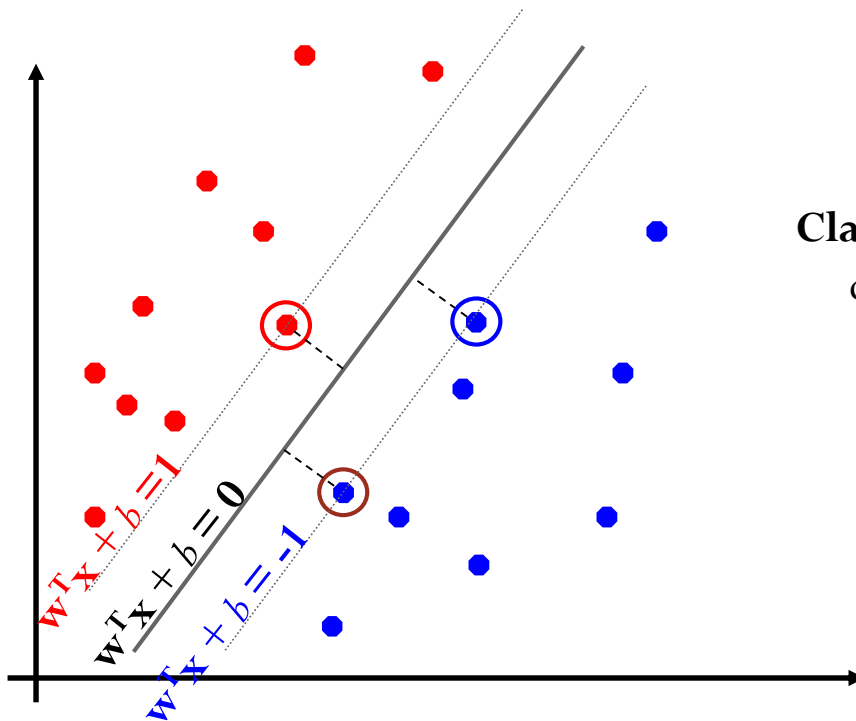
- ◆ For support vector  $\mathbf{x}_s$  the above inequality is an equality. After rescaling  $\mathbf{w}$  and  $b$  by  $\rho/2$ , we obtain that distance between each  $\mathbf{x}_s$  and the hyperplane is

$$r = \frac{y_s(\mathbf{w}^T \mathbf{x}_s + b)}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$$

# Linear SVM

- ◆ Then the margin can be expressed through (rescaled)  $\mathbf{w}$  and  $b$  as:

$$\rho = 2r = \frac{2}{\|\mathbf{w}\|}$$



**Classification rule:**

classify as: **+1** if  $\mathbf{w}^T \mathbf{x} + b \geq 1$   
**-1** if  $\mathbf{w}^T \mathbf{x} + b \leq -1$

universe explodes if  $|\mathbf{w}^T \mathbf{x} + b| < 1$

# Observations

◆ We can assume  $b=0$

**Classification rule:**

classify as: **+1**    if    $\mathbf{w}^\top \mathbf{x} + b \geq 1$

**-1**    if    $\mathbf{w}^\top \mathbf{x} + b \leq -1$

universe    if    $|\mathbf{w}^\top \mathbf{x} + b| < 1$   
explodes

◆ This is the same as:

$$y_i \mathbf{w}^\top \mathbf{x}_i \geq 1, \quad \forall i = 1, \dots, N$$



# The Primal Hard SVM

- ◆ Given training dataset:  $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$
- ◆ Assume that  $D$  is linearly separable

$$\begin{aligned}\hat{\mathbf{w}} &= \arg \min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{s.t.}: & \quad y_i \mathbf{w}^\top \mathbf{x}_i \geq 1, \quad \forall i = 1, \dots, N\end{aligned}$$

- ◆ Prediction:

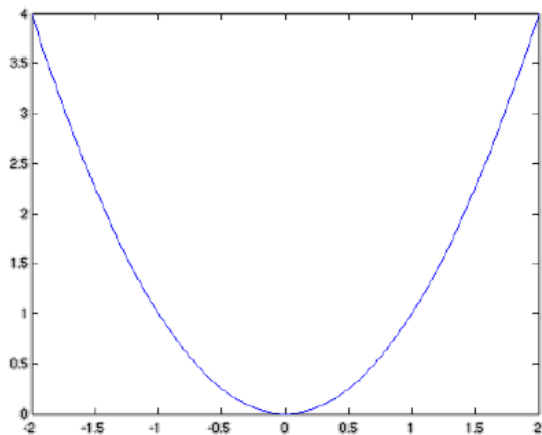
$$f(\mathbf{x}; \hat{\mathbf{w}}) = \text{sign}(\hat{\mathbf{w}}^\top \mathbf{x})$$

This is a QP problem (d-dimensional)  
(Quadratic cost function, linear constraints)

# Constrained Optimization

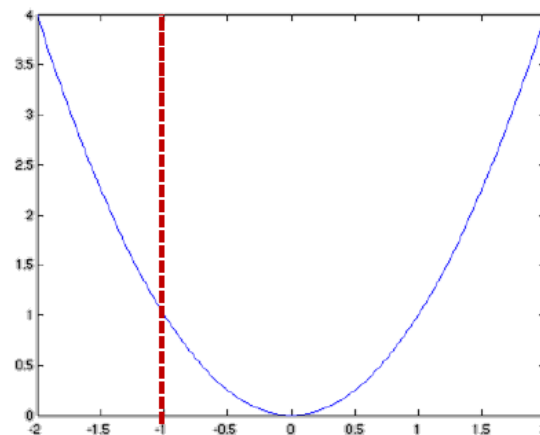
$$\begin{array}{ll}\min_x & x^2 \\ \text{s.t.} & x \geq b\end{array}$$

$$\min_x x^2$$



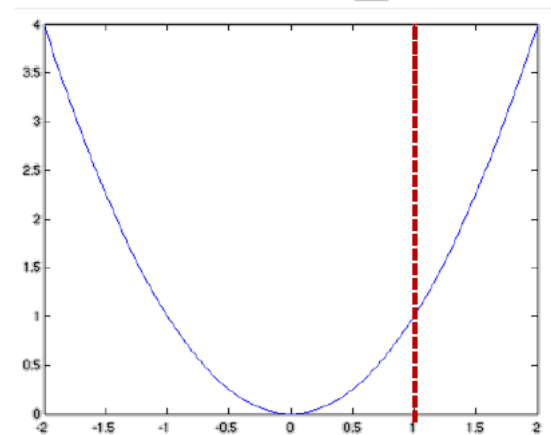
$$x^* = 0$$

$$\begin{array}{ll}\min_x & x^2 \\ \text{s.t.} & x \geq -1\end{array}$$



$$x^* = 0$$

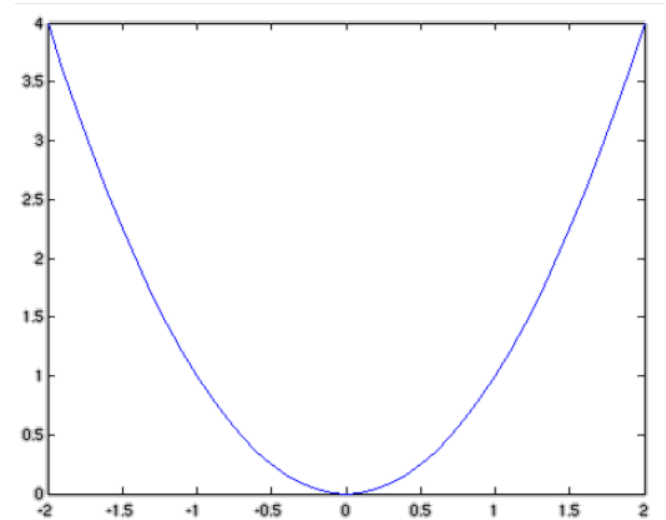
$$\begin{array}{ll}\min_x & x^2 \\ \text{s.t.} & x \geq 1\end{array}$$



$$x^* = 1$$

# Lagrange Multiplier

$$\begin{array}{ll}\min_x & x^2 \\ \text{s.t.} & x \geq b\end{array}$$



◆ Move the constraint to objective function – Lagrangian

$$L(x, \alpha) = x^2 - \alpha(x - b), \quad \text{s.t.}: \alpha \geq 0$$

◆ Solve:

$$\begin{array}{ll}\min_x \max_{\alpha} & L(x, \alpha) \\ \text{s.t.}: & \alpha \geq 0\end{array}$$

**Constraint is active when  $\alpha > 0$**

# Lagrange Multiplier – dual variables

◆ Solving:

$$\begin{array}{ll} \min_x \max_{\alpha} & L(x, \alpha) = x^2 - \alpha(x - b) \\ \text{s.t.} & \alpha \geq 0 \end{array}$$

◆ We get:

$$\frac{\partial L}{\partial x} = 0 \Rightarrow x^* = \frac{\alpha}{2}$$

$$\frac{\partial L}{\partial \alpha} = 0 \Rightarrow \alpha^* = \max(2b, 0)$$

When  $\alpha > 0$ , constraint is tight

# From Primal to Dual

◆ Primal problem:

$$\begin{aligned}\hat{\mathbf{w}} &= \arg \min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{s.t.:} \quad & y_i \mathbf{w}^\top \mathbf{x}_i \geq 1, \quad \forall i = 1, \dots, N\end{aligned}$$

◆ Lagrangian function:

$$L(\mathbf{w}, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i (y_i \mathbf{w}^\top \mathbf{x}_i - 1)$$

$$\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_N)^\top \geq 0$$

# The Lagrange Problem

$$L(\mathbf{w}, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i (y_i \mathbf{w}^\top \mathbf{x}_i - 1)$$

◆ The Lagrange problem:

$$(\hat{\mathbf{w}}, \hat{\boldsymbol{\alpha}}) = \arg \min_{\mathbf{w}} \max_{\boldsymbol{\alpha}} L(\mathbf{w}, \boldsymbol{\alpha})$$

$$0 = \frac{\partial L(\mathbf{w}, \boldsymbol{\alpha})}{\partial \mathbf{w}} \Big|_{\hat{\mathbf{w}}} = \hat{\mathbf{w}} - \sum_i \alpha_i y_i \mathbf{x}_i$$

$$\Rightarrow \hat{\mathbf{w}} = \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i$$

# The Dual Problem

$$L(\mathbf{w}, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i (y_i \mathbf{w}^\top \mathbf{x}_i - 1)$$

$$\Rightarrow \hat{\mathbf{w}} = \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i$$

$$\Rightarrow L(\hat{\mathbf{w}}, \boldsymbol{\alpha}) = \frac{1}{2} \|\hat{\mathbf{w}}\|^2 - \sum_i \alpha_i (y_i \hat{\mathbf{w}}^\top \mathbf{x}_i - 1)$$

$$= \frac{1}{2} \left\| \sum_i \alpha_i y_i \mathbf{x}_i \right\|^2 + \boldsymbol{\alpha}^\top \mathbf{1} - \sum_i \alpha_i y_i \left( \sum_j \alpha_j y_j \mathbf{x}_j \right)^\top \mathbf{x}_i$$

$$= \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2} \boldsymbol{\alpha}^\top \mathbf{Y} \mathbf{G} \mathbf{Y} \boldsymbol{\alpha}$$

$$\mathbf{Y} := \text{diag}(y_1, \dots, y_N)$$

$$\mathbf{G} \in \mathbb{R}^{N \times N}, \text{ where } G_{ij} := \mathbf{x}_i^\top \mathbf{x}_j \quad \text{Gram matrix}$$

# The Dual Hard SVM

$$\mathbf{Y} := \text{diag}(y_1, \dots, y_N)$$

$$G \in \mathbb{R}^{N \times N}, \text{ where } G_{ij} := \mathbf{x}_i^\top \mathbf{x}_j \quad \text{Gram matrix}$$

$$\begin{aligned} \hat{\boldsymbol{\alpha}} &= \arg \max_{\boldsymbol{\alpha}} \quad \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2} \boldsymbol{\alpha}^\top \mathbf{Y} G \mathbf{Y} \boldsymbol{\alpha} \\ \text{s.t.: } &\alpha_i \geq 0, \quad \forall i = 1, \dots, N \end{aligned}$$

◆ Optimal solution:

$$\hat{\mathbf{w}} = \sum_{i=1}^N \hat{\alpha}_i y_i \mathbf{x}_i$$

◆ Prediction:

$$f(\mathbf{x}; \hat{\mathbf{w}}) = \text{sign}(\hat{\mathbf{w}}^\top \mathbf{x}) = \text{sign} \left( \sum_{i=1}^N \hat{\alpha}_i y_i \mathbf{x}_i^\top \mathbf{x} \right)$$



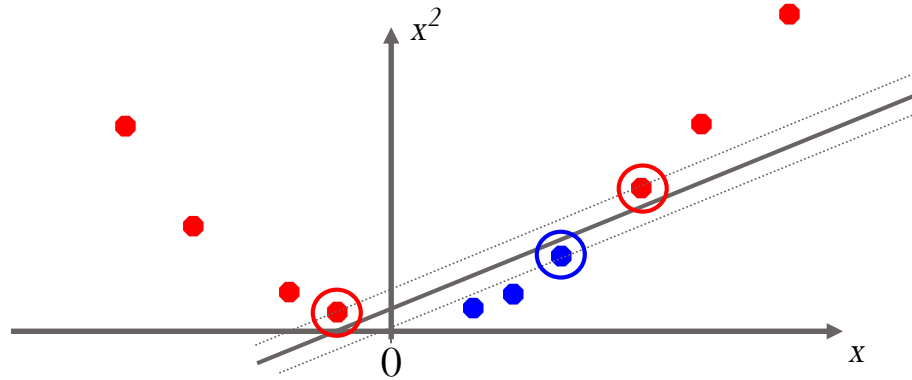
# The Problem with Hard SVM

- ◆ It assumes samples are linearly separable ...
- ◆ How about if the data is not linearly separable?



# The Problem with Hard SVM

- ◆ If the data is not linearly separable, adding new features might make it linearly separable



- Now drop this “augmented” data into our linear SVM!

# The Problem with Hard SVM

◆ It assumes samples are linearly separable

◆ Solutions:

- User feature transformation to a higher-dim space
  - Overfitting ☹
- Soft margin SVM instead of hard SVM
  - Next slides

# Hard SVM

◆ The hard SVM problem can be rewritten:

$$\begin{aligned}\hat{\mathbf{w}} &= \arg \min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{s.t.}: & \quad y_i \mathbf{w}^\top \mathbf{x}_i > 0, \quad \forall i = 1, \dots, N\end{aligned}$$



$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^N \ell_{0-\infty}(y_i \mathbf{w}^\top \mathbf{x}_i)$$

$$\text{where } \ell_{0-\infty}(b) = \begin{cases} \infty & \text{if } b < 0 \\ 0 & \text{if } b > 0 \end{cases}$$

# From Hard to Soft Constraints

- ◆ Instead of using hard constraints (**linearly separable**)

$$\hat{\mathbf{w}}_{hard} = \arg \min_{\mathbf{w}} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^N \ell_{0-\infty}(y_i \mathbf{w}^\top \mathbf{x}_i)$$

- ◆ We can try to solve the soft version of it:
  - The loss is only 1 instead of  $\infty$  if misclassify an instance

$$\hat{\mathbf{w}}_{soft} = \arg \min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^N \ell_{0-1}(y_i \mathbf{w}^\top \mathbf{x}_i)$$

$$\text{where } \ell_{0-1}(b) = \begin{cases} 1 & \text{if } b < 0 \\ 0 & \text{if } b > 0 \end{cases}$$

## Problems with 0/1 loss

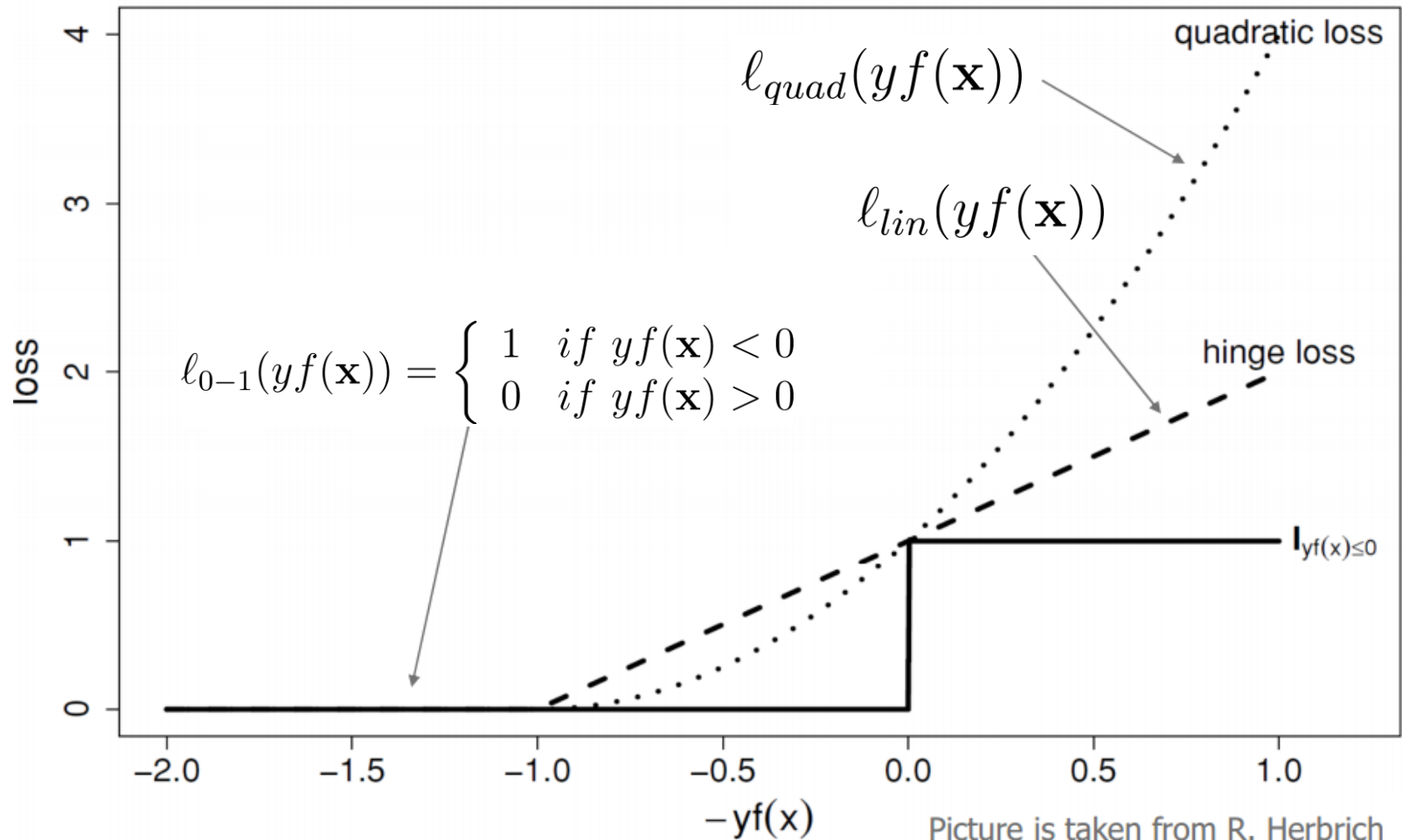
$$\hat{\mathbf{w}}_{soft} = \arg \min_{\mathbf{w}} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^N \ell_{0-1}(y_i \mathbf{w}^\top \mathbf{x}_i)$$

$$\text{where } \ell_{0-1}(b) = \begin{cases} 1 & \text{if } b < 0 \\ 0 & \text{if } b > 0 \end{cases}$$

- ◆ It is not convex in  $y\mathbf{w}^\top \mathbf{x}$ 
  - It is not convex in  $\mathbf{w}$ , either
- ◆ We like convex functions ...

# Approximation of the step function

$$f(\mathbf{x}) := \mathbf{w}^\top \mathbf{x}$$



# Approximation of 0/1 loss

- ◆ Piecewise linear approximation (hinge loss, convex, nonsmooth)

$$\ell_{lin}(yf(\mathbf{x})) = \max(0, 1 - yf(\mathbf{x}))$$

□ we want  $yf(\mathbf{x}) > 1$

- ◆ Quadratic approximation (square-loss, convex, smooth)

$$\ell_{quad}(yf(\mathbf{x})) = \max(0, 1 - yf(\mathbf{x}))^2$$

- ◆ Huber loss (combine the above two, convex, smooth)

$$\ell_{Huber}(yf(\mathbf{x})) = \begin{cases} 1 - yf(\mathbf{x}) & \text{if } yf(\mathbf{x}) < 0 \\ \max(0, 1 - yf(\mathbf{x}))^2 & \text{if } yf(\mathbf{x}) \geq 0 \end{cases}$$



# The Hinge loss approximation of 0/1 loss

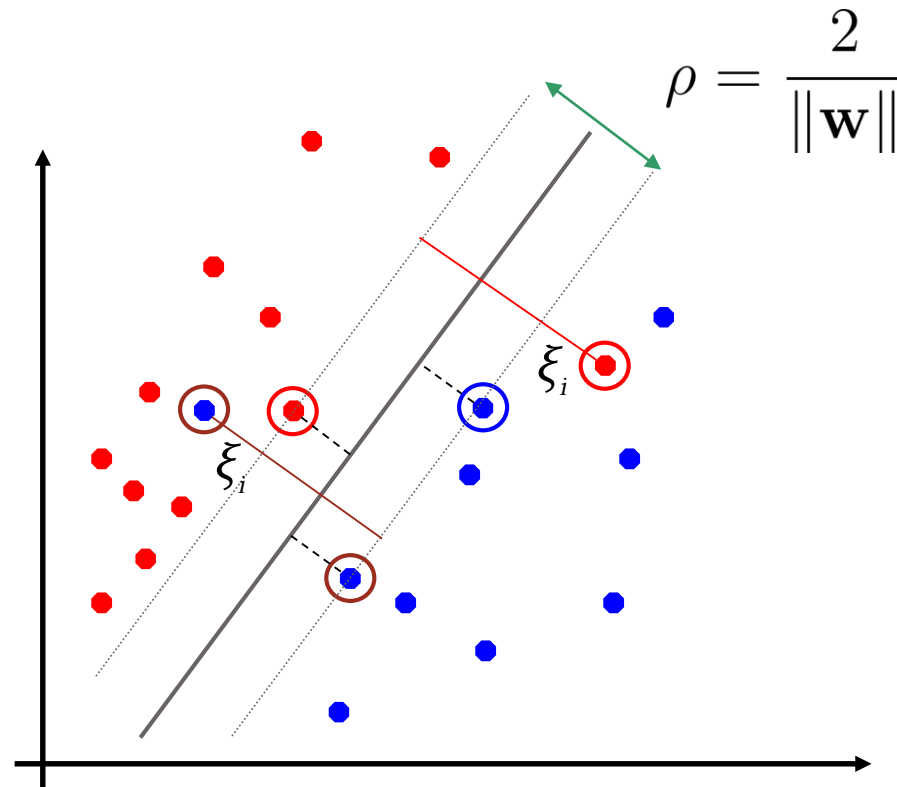
$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^N \ell_{lin}(y_i \mathbf{w}^\top \mathbf{x}_i)$$

□ where:

$$\begin{aligned} \ell_{lin}(y_i \mathbf{w}^\top \mathbf{x}_i) &= \max(0, 1 - y_i \mathbf{w}^\top \mathbf{x}_i) \\ &\geq \ell_{0-1}(y_i \mathbf{w}^\top \mathbf{x}_i) \end{aligned}$$

□ The hinge loss upper bounds the 0/1 loss

# Geometric interpretation: slack variables



$$\xi_i := \ell_{lin}(y_i \mathbf{w}^\top \mathbf{x}_i) = \max(0, 1 - y_i \mathbf{w}^\top \mathbf{x}_i)$$

# The Primal Soft SVM problem

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^N \xi_i$$

where  $\xi_i := \ell_{lin}(y_i \mathbf{w}^\top \mathbf{x}_i) = \max(0, 1 - y_i \mathbf{w}^\top \mathbf{x}_i)$

◆ Equivalently:

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}, \xi} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^N \xi_i$$

$$\text{s.t.: } y_i \mathbf{w}^\top \mathbf{x}_i \geq 1 - \xi_i, \quad \forall i = 1, \dots, N$$

$$\xi_i \geq 0, \quad \forall i = 1, \dots, N$$

# The Primal Soft SVM problem

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}, \xi} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^N \xi_i$$

$$\text{s.t.: } y_i \mathbf{w}^\top \mathbf{x}_i \geq 1 - \xi_i, \quad \forall i = 1, \dots, N$$

$$\xi_i \geq 0, \quad \forall i = 1, \dots, N$$

◆ Equivalently:

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}, \xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i$$

$$C = \frac{1}{\lambda}$$

## Dual Soft SVM (using hinge loss)

$$\begin{aligned}\hat{\mathbf{w}} &= \arg \min_{\mathbf{w}, \boldsymbol{\xi}} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i \\ \text{s.t.: } & y_i \mathbf{w}^\top \mathbf{x}_i \geq 1 - \xi_i, \quad \forall i = 1, \dots, N \\ & \xi_i \geq 0, \quad \forall i = 1, \dots, N\end{aligned}$$

◆ Lagrange multipliers

$$\boldsymbol{\alpha} \geq 0, \quad \boldsymbol{\beta} \geq 0$$

◆ Lagrangian function

$$L(\mathbf{w}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \xi_i - \sum_i \alpha_i (y_i \mathbf{w}^\top \mathbf{x}_i - 1 + \xi_i) - \sum_i \beta_i \xi_i$$

$$\min_{\mathbf{w}, \boldsymbol{\xi}} \max_{\boldsymbol{\alpha}, \boldsymbol{\beta}} L(\mathbf{w}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})$$

## Dual Soft SVM (using hinge loss)

$$L(\mathbf{w}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2} \|\mathbf{w}\|^2 + C \boldsymbol{\xi}^\top \mathbf{1} - \sum_i \alpha_i y_i \mathbf{w}^\top \mathbf{x}_i + \boldsymbol{\alpha}^\top \mathbf{1} - \boldsymbol{\xi}^\top (\boldsymbol{\alpha} + \boldsymbol{\beta})$$

◆ We get:

$$0 = \frac{\partial L}{\partial \mathbf{w}} \Big|_{\hat{\mathbf{w}}} \quad \Rightarrow \quad \hat{\mathbf{w}} = \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i$$

$$0 = \frac{\partial L}{\partial \boldsymbol{\xi}} \Big|_{\hat{\boldsymbol{\xi}}} \quad \Rightarrow \quad \boldsymbol{\beta} = C \mathbf{1} - \boldsymbol{\alpha} \geq 0$$

$$\Rightarrow 0 \leq \boldsymbol{\alpha} \leq C \mathbf{1}$$

◆ Dual problem:

$$(\hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}}) = \underset{0 \leq \boldsymbol{\alpha} \leq C \mathbf{1}; 0 \leq \boldsymbol{\beta}}{\operatorname{argmax}} L(\hat{\mathbf{w}}, \hat{\boldsymbol{\xi}}, \boldsymbol{\alpha}, \boldsymbol{\beta})$$

$$\hat{\boldsymbol{\alpha}} = \underset{0 \leq \boldsymbol{\alpha} \leq C \mathbf{1}}{\operatorname{argmax}} \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2} \boldsymbol{\alpha}^\top \mathbf{Y} G \mathbf{Y} \boldsymbol{\alpha}$$

## Dual Soft SVM (using hinge loss)

$$\mathbf{Y} := \text{diag}(y_1, \dots, y_N)$$

$$G \in \mathbb{R}^{N \times N}, \text{ where } G_{ij} := \mathbf{x}_i^\top \mathbf{x}_j \quad \text{Gram matrix}$$

$$\hat{\boldsymbol{\alpha}} = \underset{0 \leq \boldsymbol{\alpha} \leq C\mathbf{1}}{\operatorname{argmax}} \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2} \boldsymbol{\alpha}^\top \mathbf{Y} G \mathbf{Y} \boldsymbol{\alpha}$$

- ◆ This is the same as the dual hard SVM problem, except that we have additional constraints

# SVM in the dual space

◆ Solve the dual problem

$$\hat{\boldsymbol{\alpha}} = \underset{0 \leq \boldsymbol{\alpha} \leq C\mathbf{1}}{\operatorname{argmax}} \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2} \boldsymbol{\alpha}^\top \mathbf{Y} \mathbf{G} \mathbf{Y} \boldsymbol{\alpha}$$

◆ The primal solution

$$\Rightarrow \hat{\mathbf{w}} = \sum_{i=1}^N \hat{\alpha}_i y_i \mathbf{x}_i$$

◆ Prediction

$$f(\mathbf{x}; \hat{\mathbf{w}}) = \operatorname{sign}(\hat{\mathbf{w}}^\top \mathbf{x}) = \operatorname{sign}\left(\sum_{i=1}^N \hat{\alpha}_i y_i \mathbf{x}_i^\top \mathbf{x}\right)$$



# Why it is called Support Vector Machines?

◆ Hard-SVM:


$$L(\mathbf{w}, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i (y_i \mathbf{w}^\top \mathbf{x}_i - 1)$$

$$\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_N)^\top \geq 0$$

◆ KKT conditions (**complementary slackness condition**):

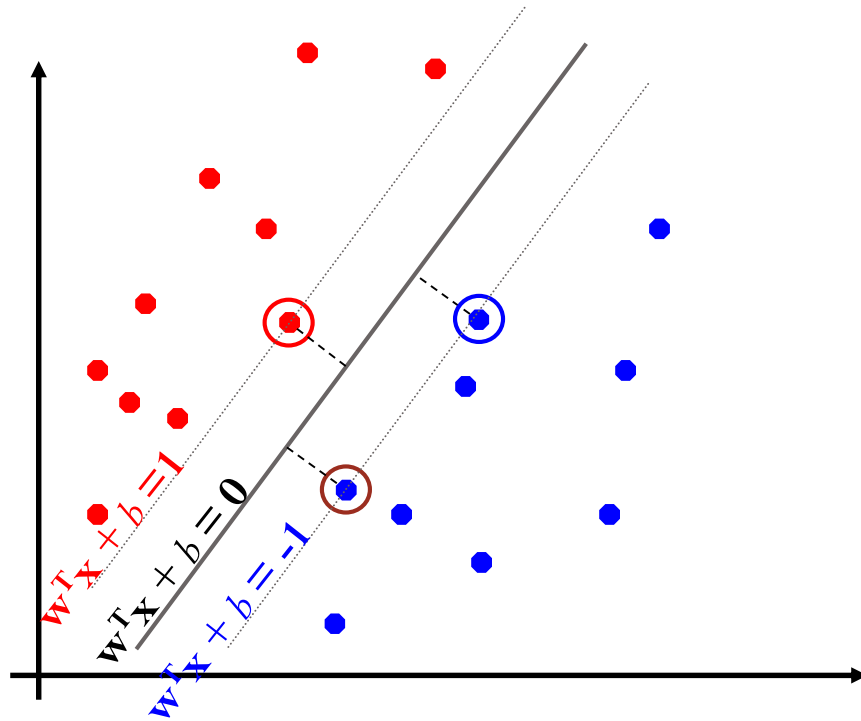
$$\forall i : \hat{\alpha}_i (y_i \hat{\mathbf{w}}^\top \mathbf{x}_i - 1) = 0$$

$$\hat{\alpha}_i = 0 \quad OR \quad \hat{\alpha}_i > 0 \Rightarrow y_i \hat{\mathbf{w}}^\top \mathbf{x}_i = 1$$

  $\mathbf{x}_i$  is on the margin line! **SUPPORT VECTORS**

# Why it is called Support Vector Machines?

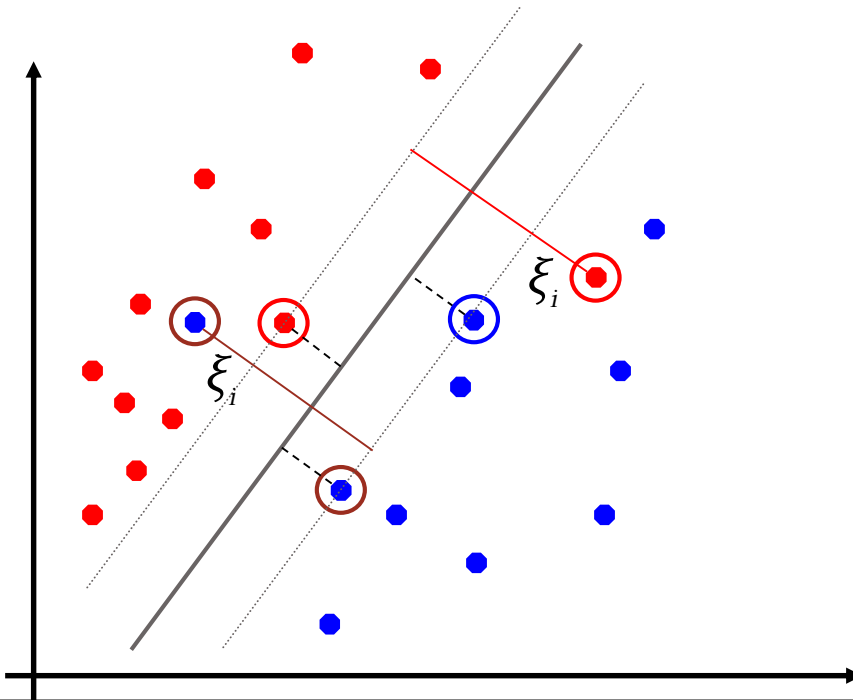
◆ Hard SVM:



- Only need to store support vectors to predict labels of test data

# Support vectors in Soft SVM

$$\begin{aligned}\hat{\mathbf{w}} &= \arg \min_{\mathbf{w}, \xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i \\ \text{s.t.: } & y_i \mathbf{w}^\top \mathbf{x}_i \geq 1 - \xi_i, \quad \forall i = 1, \dots, N \\ & \xi_i \geq 0, \quad \forall i = 1, \dots, N\end{aligned}$$



◆ Margin support vectors

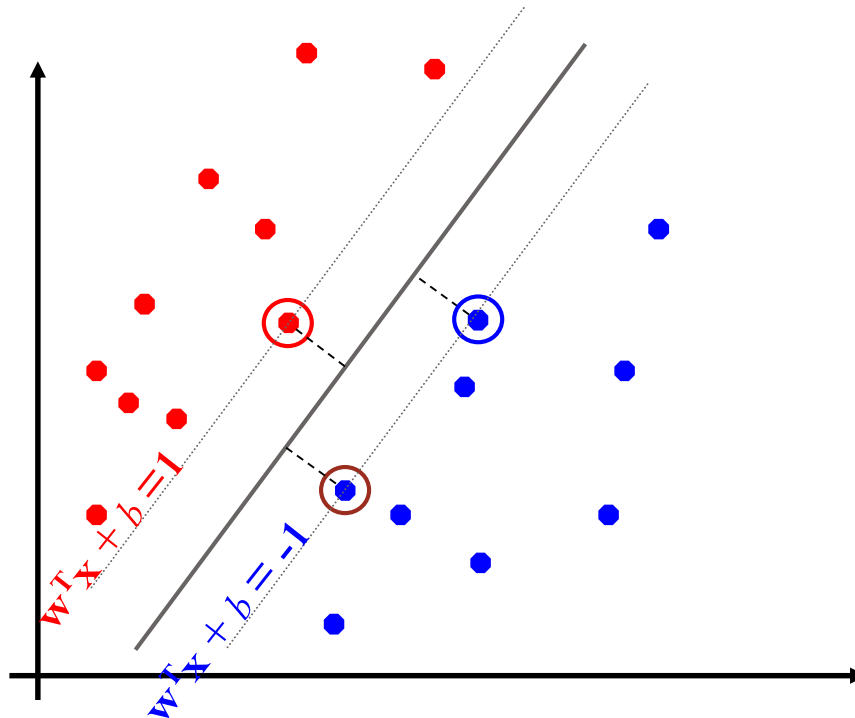
$$y_i \mathbf{w}^\top \mathbf{x}_i = 1$$

◆ Nonmargin support vectors

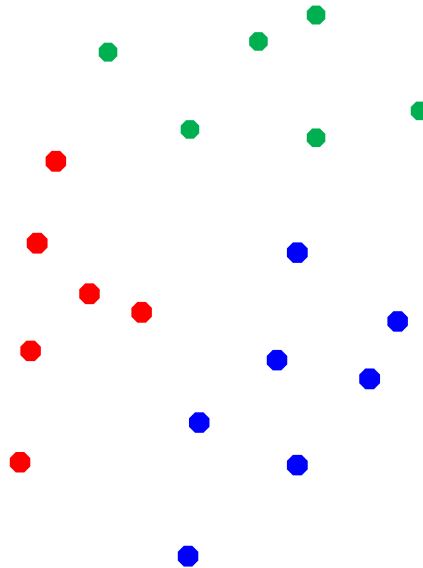
$$\xi_i > 0$$

# Dual Sparsity

- ◆ Only few Lagrange multipliers (dual variables)  $\alpha_i$  can be non-zero



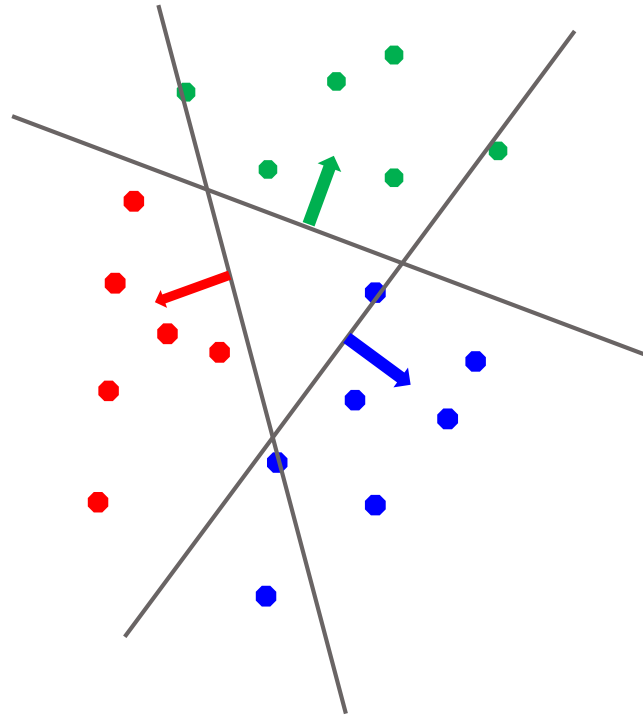
# What about multiple classes?



# One vs All

◆ Learn multiple binary classifiers separately:

□ class  $k$  vs. rest  $(\mathbf{w}_k, b_k)_{k=1,2,3}$



◆ Prediction:

$$\hat{y} = \underset{k}{\operatorname{argmax}} (\mathbf{w}_k^\top \mathbf{x} + b_k)$$

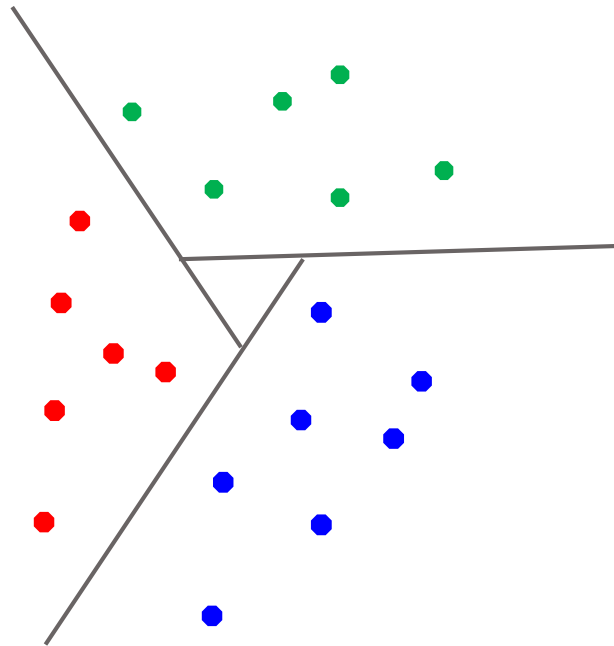
# Problems with One vs All?

$$\hat{y} = \operatorname{argmax}_k (\mathbf{w}_k^\top \mathbf{x} + b_k)$$

- ◆ (1) The weights may not be based on the same scale
  - Note:  $(a\mathbf{w}_k)^\top \mathbf{x} + (ab_k)$  is also a solution
  
- ◆ (2) Imbalance issue when learning each binary classifier
  - Much more negatives than positives

# One vs One

◆ Learn  $K(K-1)/2$  binary classifiers



◆ Prediction:

□ Majority voting

◆ Ambiguity issue!



# Learning 1 Joint Classifier

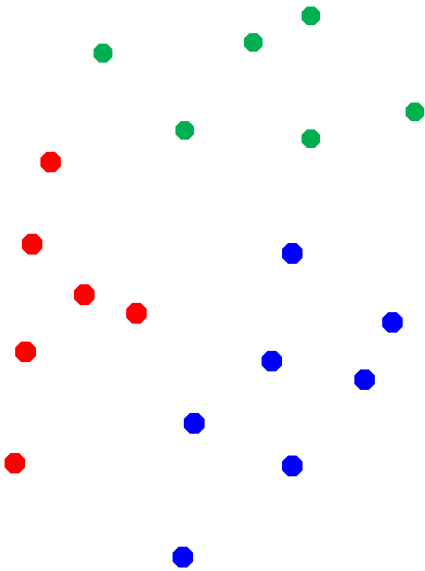
- ◆ Simultaneously learn 3 sets of weights  $(\mathbf{w}_k, b_k)_{k=1,2,3}$

$$\hat{y} = \operatorname{argmax}_k (\mathbf{w}_k^\top \mathbf{x} + b_k)$$

$$\forall i, \forall y \neq y_i :$$

$$\mathbf{w}_{y_i}^\top \mathbf{x}_i + b_{y_i} \geq \mathbf{w}_y^\top \mathbf{x}_i + b_y + 1$$

**Margin:** gap between true class and nearest other class



# Learning 1 Joint Classifier

- ◆ Simultaneously learn 3 sets of weights  $(\mathbf{w}_k, b_k)_{k=1,2,3}$

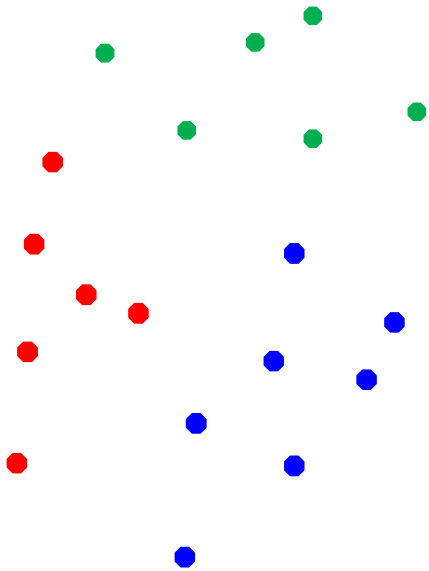
Joint optimization:

$$\min_{\mathbf{w}, b} \frac{1}{2} \sum_y \|\mathbf{w}_y\|^2 + C \sum_{i=1}^N \sum_{y \neq y_i} \xi_{iy}$$

$$\begin{aligned} \text{s.t.: } \mathbf{w}_{y_i}^\top \mathbf{x}_i + b_{y_i} &\geq \mathbf{w}_y^\top \mathbf{x}_i + b_y + 1 - \xi_{iy} \quad \forall i, \quad \forall y \neq y_i \\ \xi_{iy} &\geq 0 \quad \forall i, \quad \forall y \neq y_i \end{aligned}$$

Prediction:

$$\hat{y} = \operatorname{argmax}_k (\mathbf{w}_k^\top \mathbf{x} + b_k)$$



# What you need to know

- ◆ Maximizing margin
- ◆ Derivation of SVM formulation
- ◆ Slack variables and hinge loss
- ◆ Relationship between
  - 0/1 loss
  - Hinge loss
- ◆ Tackling multiple class
  - One vs. All
  - Multiclass SVMs

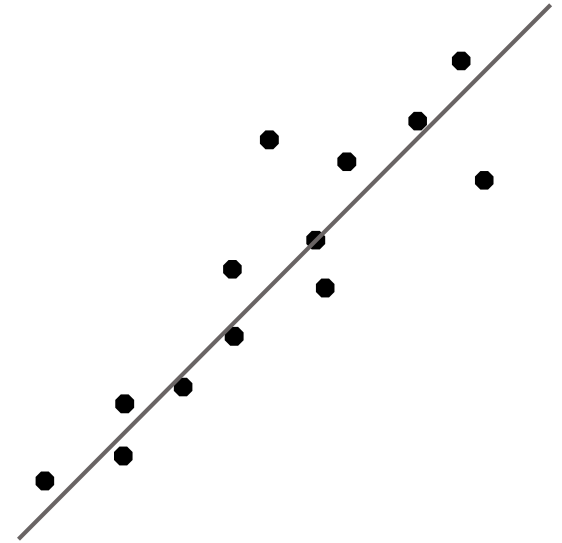
# SVM for Regression

- ◆ Training data  $(\mathbf{x}_i, y_i)$ ,  $\mathbf{x}_i \in \mathbb{R}^d$ ,  $y_i \in \mathbb{R}$
- ◆ Still learn a hyper-plane (linear model)
- ◆ Squared error is the popular loss

$$\sum_{i=1}^N (y_i - \mathbf{w}^\top \mathbf{x}_i)^2$$

- a smooth function – no sparsity
- ◆ A piecewise linear approximation ( $\epsilon$ -insensitive loss)

$$\sum_{i=1}^N \max(0, |y_i - \mathbf{w}^\top \mathbf{x}_i| - \epsilon)$$



# SVM in the dual space

◆ Without offset  $b$ :

$$\begin{aligned}\hat{\alpha} &= \operatorname{argmax}_{\alpha} \alpha^{\top} \mathbf{1} - \frac{1}{2} \alpha^{\top} \mathbf{Y} G \mathbf{Y} \alpha \\ \text{s.t.: } &0 \leq \alpha \leq C \mathbf{1}\end{aligned}$$

◆ With offset  $b$ :

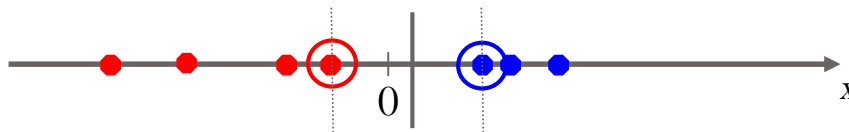
$$\begin{aligned}\hat{\alpha} &= \operatorname{argmax}_{\alpha} \alpha^{\top} \mathbf{1} - \frac{1}{2} \alpha^{\top} \mathbf{Y} G \mathbf{Y} \alpha \\ \text{s.t.: } &0 \leq \alpha \leq C \mathbf{1} \\ &\sum_i \alpha_i y_i = 0\end{aligned}$$

# Why solve the dual SVM?

- ◆ The dual problem has simpler constraints
- ◆ There some quadratic programming algorithms that can solve the dual fast, especially in high-dimensions ( $d \gg N$ )
  - See [Bottou & Lin, 2007] for a summary of dual SVM solvers
  - Be aware of the fast algorithms directly solving the primal problem, e.g., cutting-plane, stochastic subgradient, etc.
- ◆ More importantly, the **Kernel Trick**!!

# Nonlinear SVM

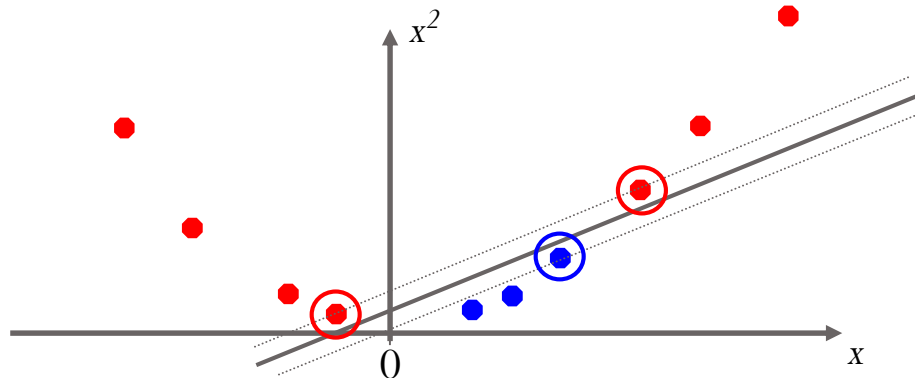
- ◆ Datasets that are linearly separable with some noise work out great:



- ◆ But what are we going to do if the dataset is just too hard?

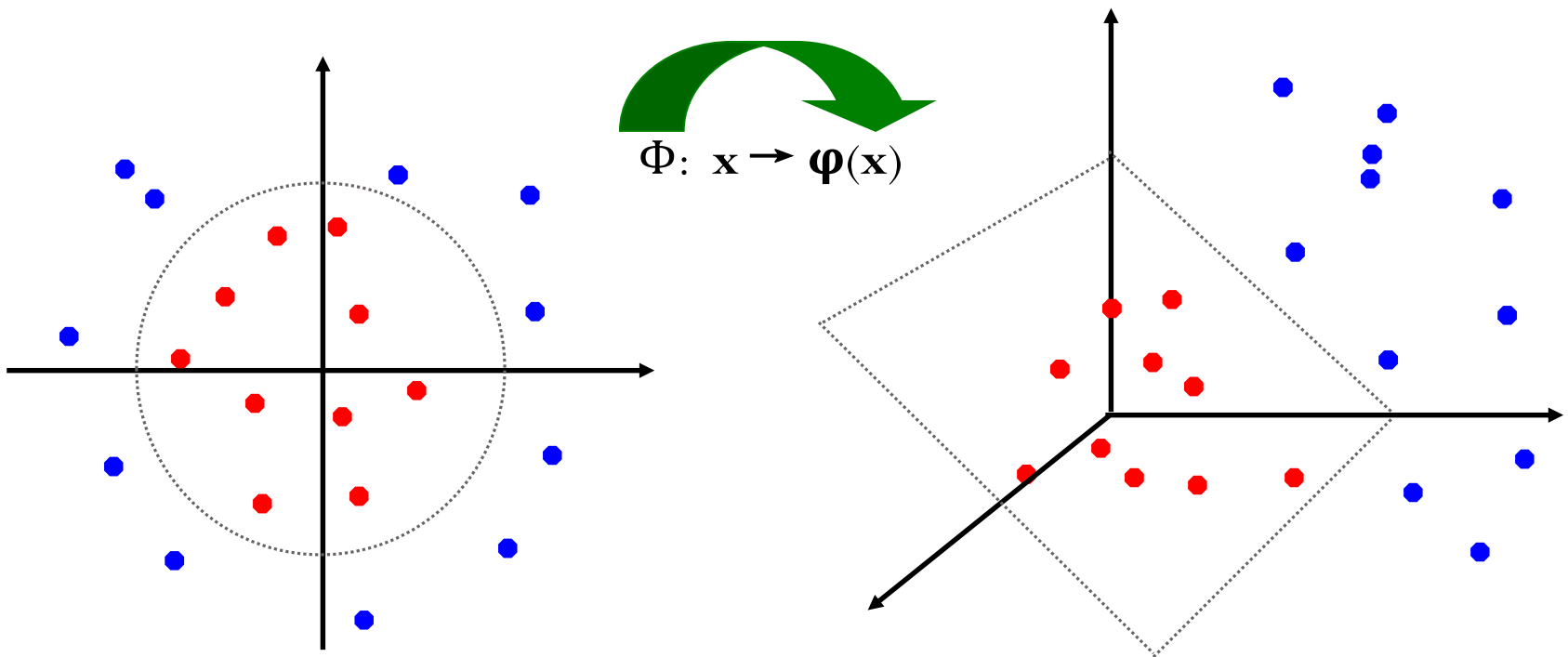


- ◆ How about... mapping data to a higher-dimensional space:



# Non-linear SVMs: Feature Spaces

- ◆ **General idea:** the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:





# Dot Product of Polynomials

◆ Polynomials of degree exactly d:  $\Phi(\mathbf{x})$

$$\mathbf{x} = (x_1, x_2)^\top, \quad \mathbf{z} = (z_1, z_2)^\top$$

◆ d=1:

$$\Phi(\mathbf{x})^\top \Phi(\mathbf{z}) = \mathbf{x}^\top \mathbf{z}$$

◆ d=2:  $\Phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)^\top$

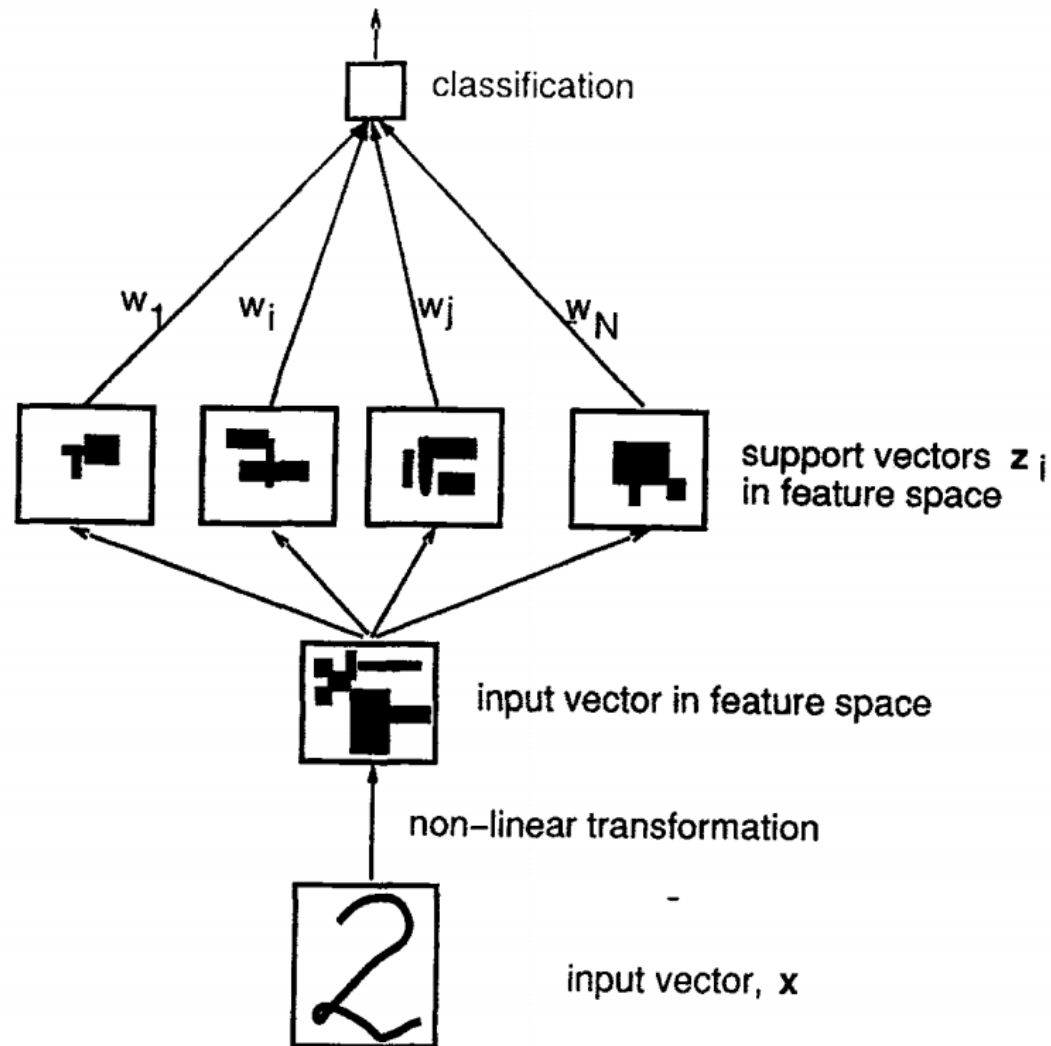
$$\Phi(\mathbf{x})^\top \Phi(\mathbf{z}) = (\mathbf{x}^\top \mathbf{z})^2$$

◆ In general:

$$\Phi(\mathbf{x})^\top \Phi(\mathbf{z}) = (\mathbf{x}^\top \mathbf{z})^d = K(\mathbf{x}, \mathbf{z})$$

# Support Vector Nets

[Cortes & Vapnik, 1995]



# The Kernel Trick

- ◆ Linear SVM relies on inner product between vectors

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j$$

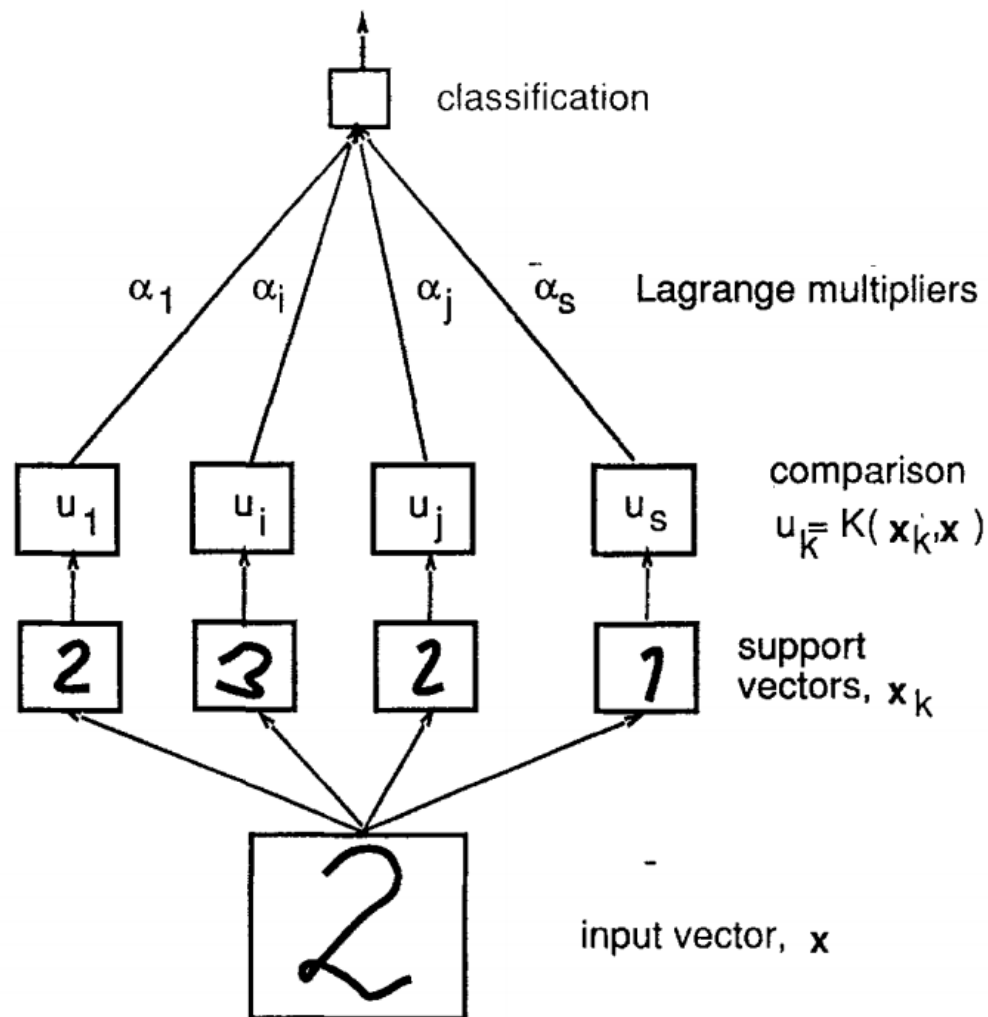
- ◆ If map every data point into high-dimensional space via  $\Phi$ :  $\mathbf{x} \rightarrow \Phi(\mathbf{x})$ , the inner product becomes:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}_j)$$

- ◆ A *kernel function* is a function that is equivalent to an inner product in some feature space.
- ◆ The feature mapping is not explicitly needed as long as we can compute the dot product using some Kernel  $K$

# Support Vector Nets

[Cortes & Vapnik, 1995]



# What functions are kernels?

◆ For some function  $K(\mathbf{x}_i, \mathbf{x}_j)$  checking that  $K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\varphi}(\mathbf{x}_i)^T \boldsymbol{\varphi}(\mathbf{x}_j)$  can be cumbersome.

◆ Mercer's theorem:

*Every semi-positive definite symmetric function is a kernel*

◆ Semi-positive definite symmetric functions correspond to a semi-positive definite symmetric Gram matrix:

$K =$

$K(\mathbf{x}_1, \mathbf{x}_1)$	$K(\mathbf{x}_1, \mathbf{x}_2)$	$K(\mathbf{x}_1, \mathbf{x}_3)$	...	$K(\mathbf{x}_1, \mathbf{x}_n)$
$K(\mathbf{x}_2, \mathbf{x}_1)$	$K(\mathbf{x}_2, \mathbf{x}_2)$	$K(\mathbf{x}_2, \mathbf{x}_3)$		$K(\mathbf{x}_2, \mathbf{x}_n)$
...	...	...	...	...
$K(\mathbf{x}_n, \mathbf{x}_1)$	$K(\mathbf{x}_n, \mathbf{x}_2)$	$K(\mathbf{x}_n, \mathbf{x}_3)$	...	$K(\mathbf{x}_n, \mathbf{x}_n)$

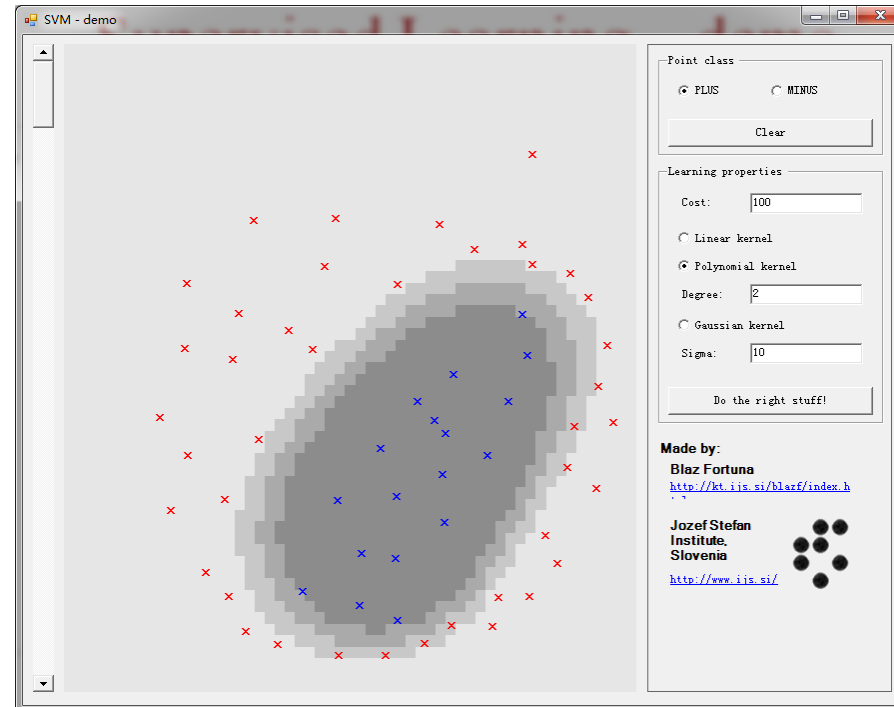
# Example Kernel Functions

- ◆ Linear:  $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$ 
  - Mapping  $\Phi: \mathbf{x} \rightarrow \Phi(\mathbf{x})$ , where  $\Phi(\mathbf{x})$  is  $\mathbf{x}$  itself
- ◆ Polynomial of power  $p$ :  $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$ 
  - Mapping  $\Phi: \mathbf{x} \rightarrow \Phi(\mathbf{x})$ , where  $\Phi(\mathbf{x})$  has  $\binom{d+p}{p}$  dimensions
- ◆ Gaussian (radial-basis function):
$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$
  - Mapping  $\Phi: \mathbf{x} \rightarrow \Phi(\mathbf{x})$ , where  $\Phi(\mathbf{x})$  is *infinite-dimensional*: every point is mapped to a *function*; combination of functions for support vectors is the separator.
- ◆ Higher-dimensional space still has *intrinsic* dimensionality  $d$ , but linear separators in it correspond to *non-linear* separators in original space.

# Overfitting

- ◆ Huge feature space with kernels, what about overfitting??
  - Maximizing margin leads to a sparse set of support vectors
  - Some interesting theory says that SVMs search for simply hypothesis with a large margin
  - Often robust to overfitting

# SVM – demo



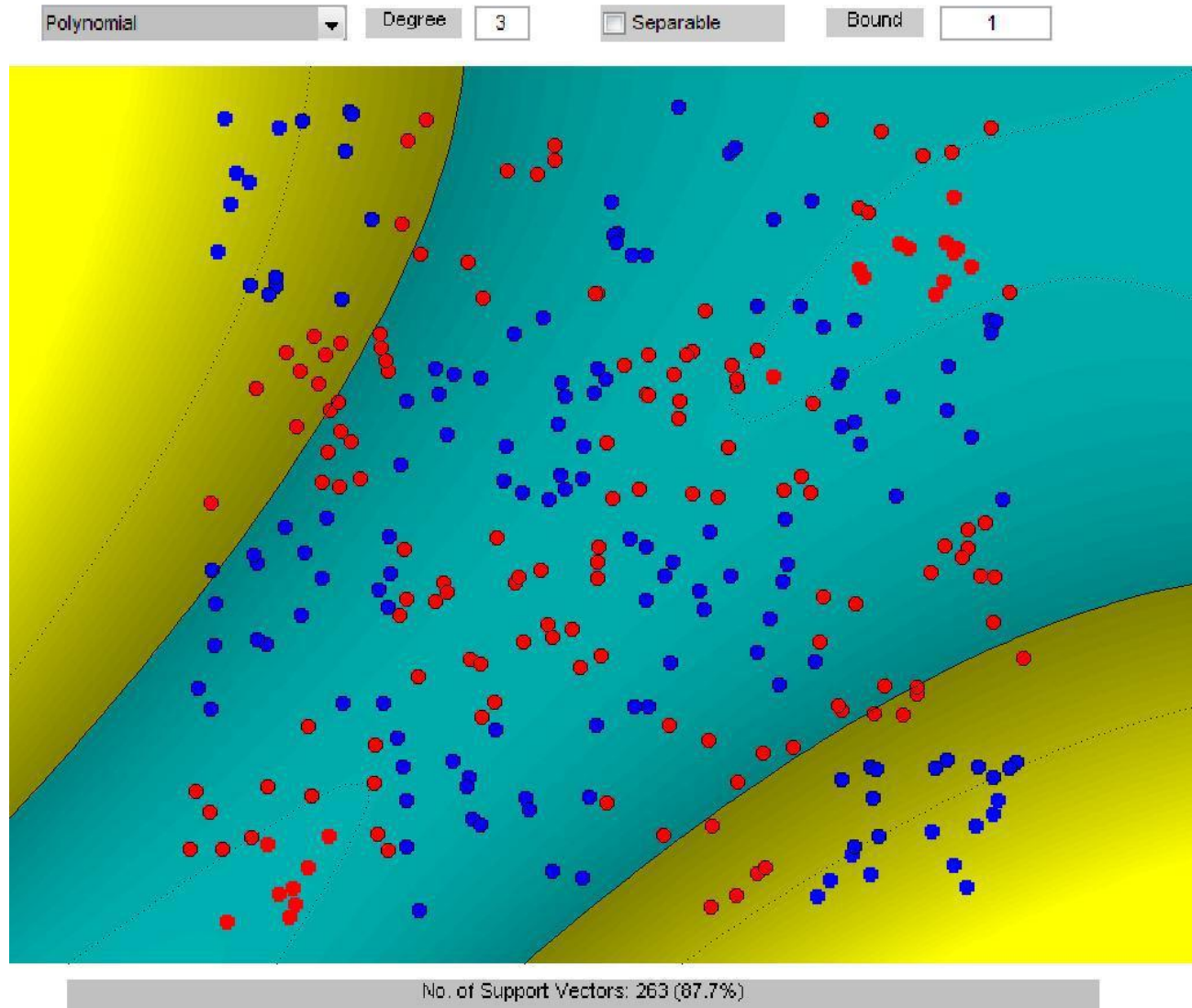
◆ <http://www.isis.ecs.soton.ac.uk/resources/svminfo/>

Good ToolKits: [1] SVM-Light: <http://svmlight.joachims.org/>

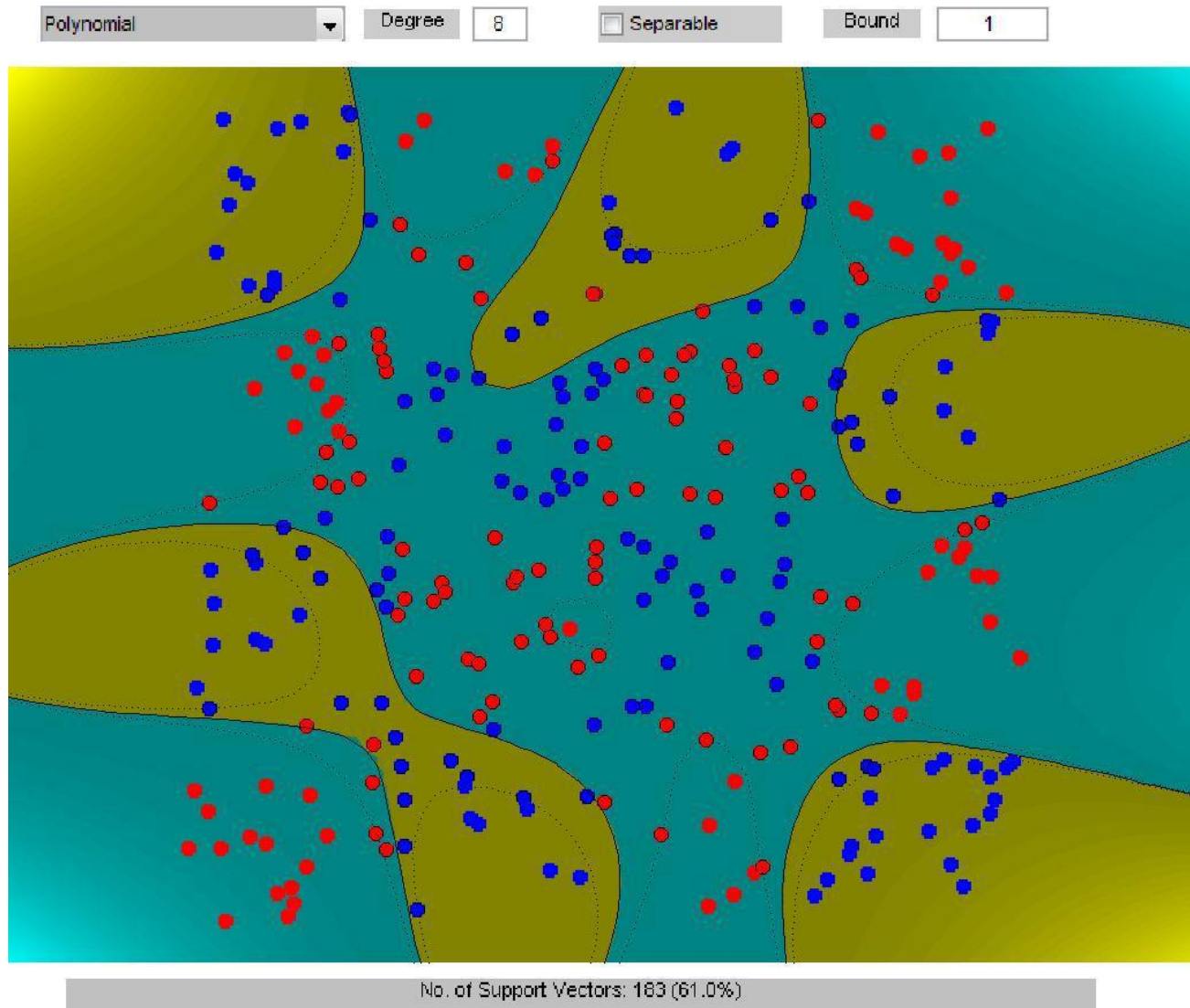
[2] LibSVM: <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>



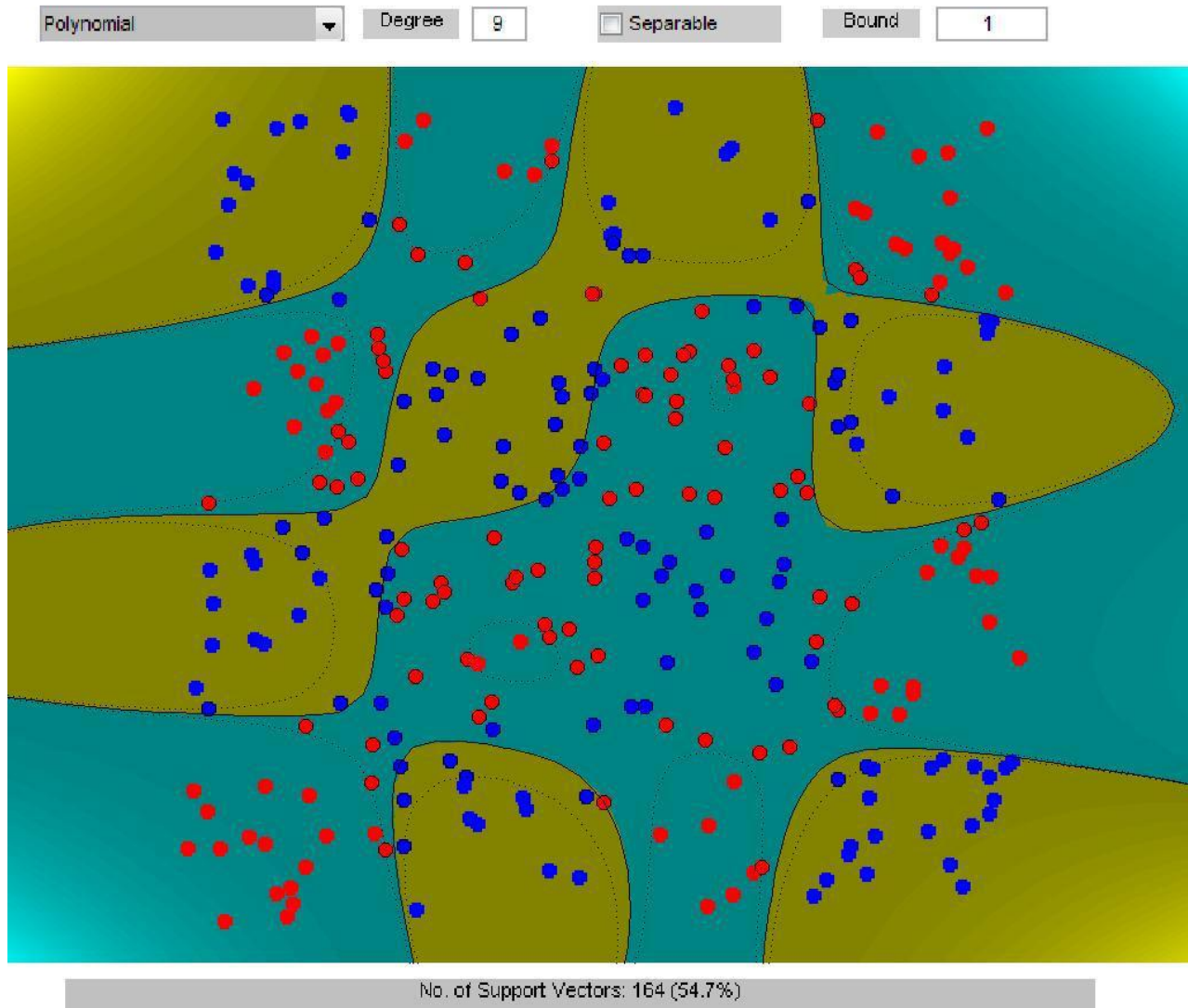
# Chessboard dataset, Polynomial kernel



# Chessboard dataset, Polynomial kernel

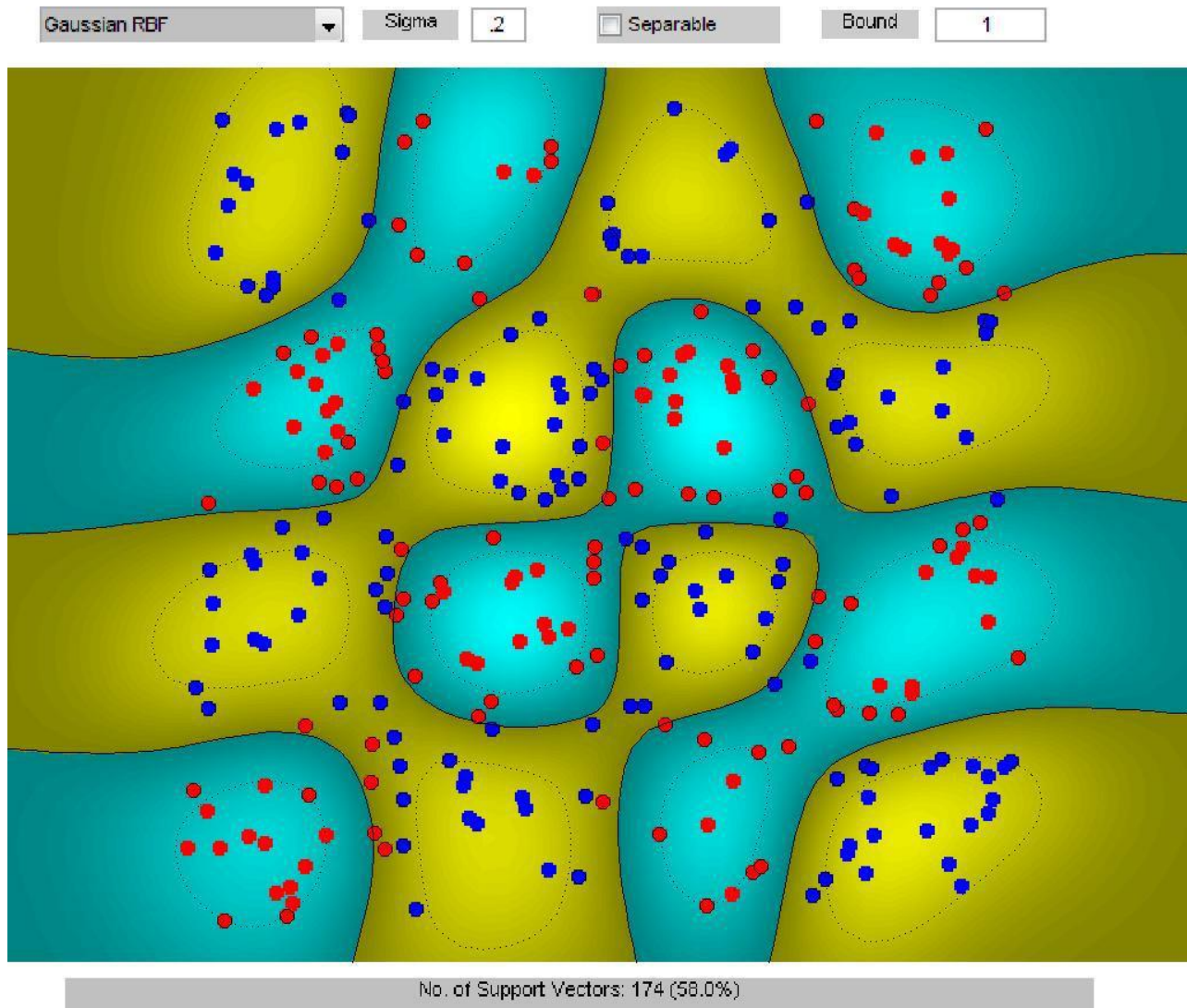


# Chessboard dataset, Polynomial kernel





# Chessboard dataset, RBF kernel



# Advanced topics

## ◆ Scalable algorithms to learn SVMs

### □ Linear SVMs

- Linear algorithm, e.g., cutting-plane (2009)
- Stochastic optimization, e.g., Pegasos (2007)
- Distributed learning, e.g., divide-and-conquer (2013)

### □ Non-linear SVMs

- Kernel approximation, e.g., using low-rank or random features

## ◆ Structured output learning with SVMs

- May cover later

◆ An incomplete list of SVM solvers [Menon, 2010]

Algorithm	Citation	SVM type	Optimization type	Style	Runtime
SMO	[Platt, 1999]	Kernel	Dual QP	Batch	$\Omega(n^2 d)$
SVM <sup>light</sup>	[Joachims, 1999]	Kernel	Dual QP	Batch	$\Omega(n^2 d)$
Core Vector Machine	[Tsang et al., 2005, 2007]	SL Kernel	Dual geometry	Batch	$O(s/\rho^4)$
SVM <sup>perf</sup>	[Joachims, 2006]	Linear	Dual QP	Batch	$O(ns/\lambda\rho^2)$
NORMA	[Kivinen et al., 2004]	Kernel	Primal SGD	Online(-style)	$\tilde{O}(s/\rho^2)$
SVM-SGD	[Bottou, 2007]	Linear	Primal SGD	Online-style	Unknown
Pegasos	[Shalev-Shwartz et al., 2007]	Kernel	Primal SGD/SGP	Online-style	$\tilde{O}(s/\lambda\rho)$
LibLinear	[Hsieh et al., 2008]	Linear	Dual coordinate descent	Batch	$O(nd \cdot \log(1/\rho))$
SGD-QN	[Bordes and Bottou, 2008]	Linear	Primal 2SGD	Online-style	Unknown
FOLOS	[Duchi and Singer, 2008]	Linear	Primal SGP	Online-style	$\tilde{O}(s/\lambda\rho)$
BMRM	[Smola et al., 2007]	Linear	Dual QP	Batch	$O(d/\lambda\rho)$
OCAS	[Franc and Sonnenburg, 2008]	Linear	Primal QP	Batch	$O(nd)$

# Validation

## ◆ Model selection:

- Almost invariably, all ML methods have some free parameters
  - The number of neighbors in K-NN
  - The kernel parameters in SVMs

## ◆ Performance estimation:

- Once we have chosen a model, how to estimate its performance?

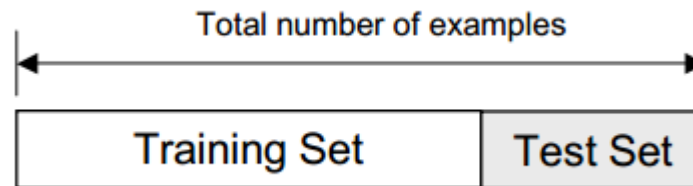
# Motivation

- ◆ If we had access to an unlimited number of examples, there is a straightforward answer
  - Choose the model with the lowest error rate on the entire population
  - The error rate is the true error rate
- ◆ In practice, we only access to a finite set of examples, usually smaller than we wanted
  - Use all training data to select model  $\Rightarrow$  too optimistic!
  - A better approach is to split the training set into disjoint subsets

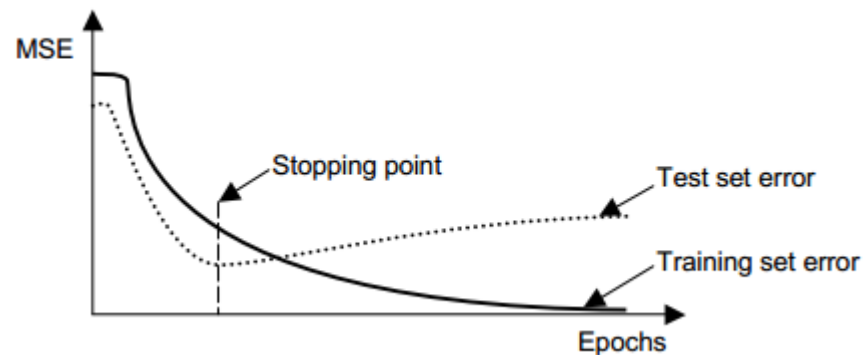


# Holdout Method

- ◆ Split dataset into two subsets
  - Training set: used to learn the classifier
  - Test set: used to estimate the error rate of the trained classifier



- ◆ E.g.: used to determine a stopping point of an iterative alg.:



# Holdout Method

## ◆ Two basic drawbacks

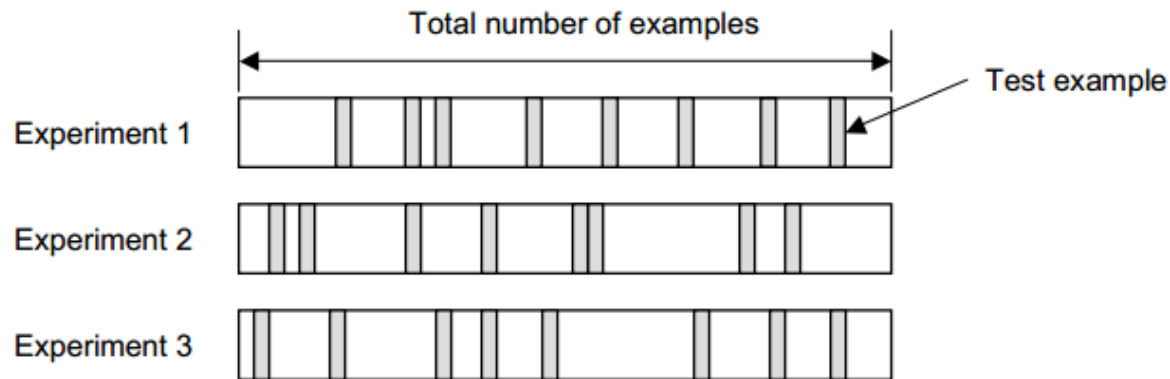
- In problems with a sparse dataset, we may not be able to afford the “luxury” of setting aside a portion of data for testing
- A single train-test split may lead to misleading results, e.g., if we happened to get an “unfortunate” split

## ◆ Resampling can overcome the limitations, but at the expense of more computations

- Cross-validation
  - Random subsampling
  - K-fold cross-validation
  - Leave-one-out cross-validation
- Bootstrap

# Random Subsampling

- ◆ Performs  $K$  data splits of the entire dataset
  - Each split randomly selects a (fixed) no. examples
  - For each split, retrain the classifier with training data, and evaluate on test examples

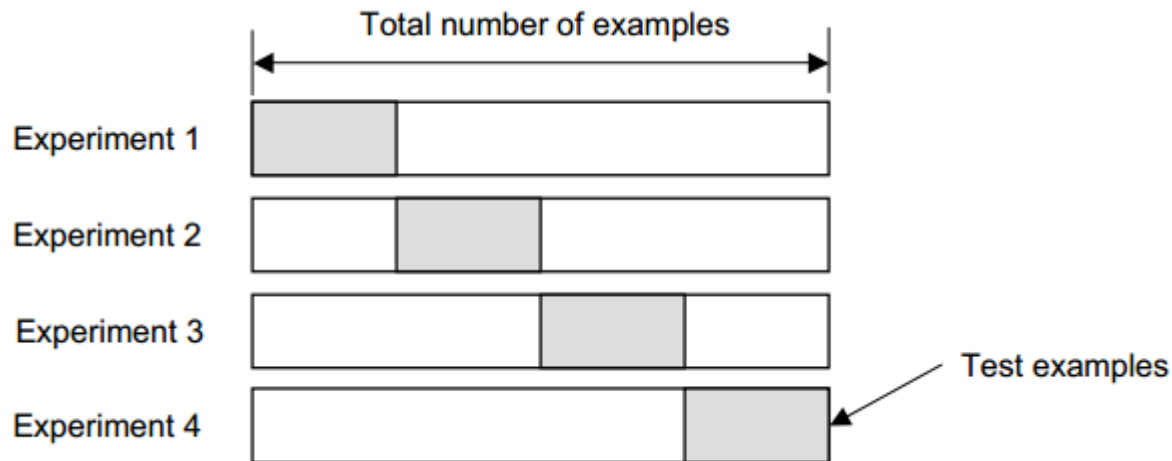


- ◆ The true error is estimated as the average

$$E = \frac{1}{K} \sum_{k=1}^K E_k$$

# K-Fold Cross-validation

- ◆ Create a K-fold partition of the dataset
  - For each of K experiments, use K-1 folds for training and the remaining one for testing

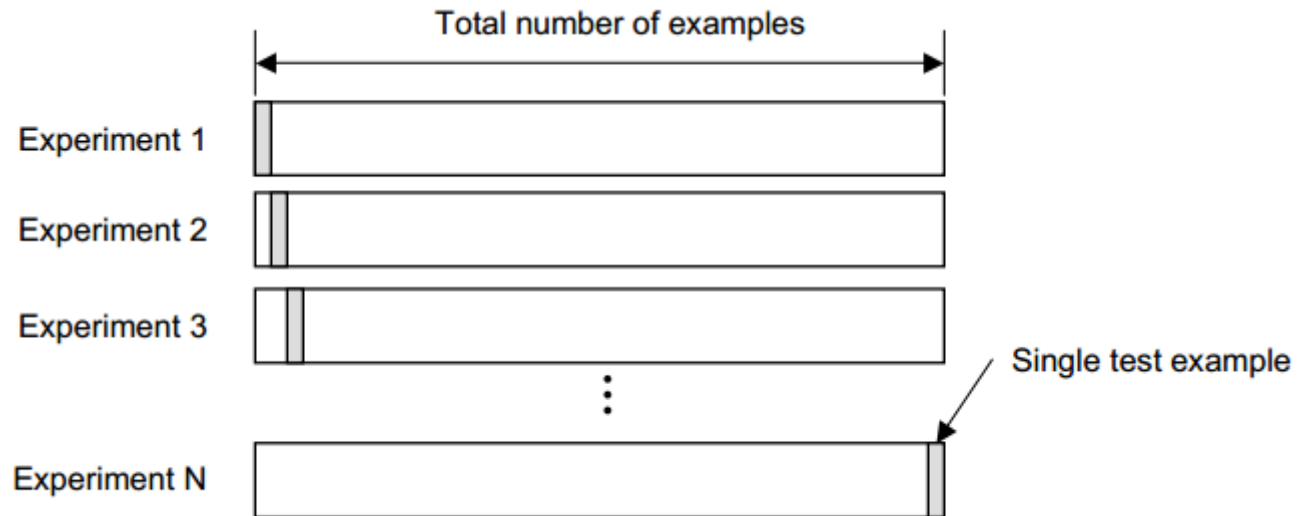


- ◆ K-fold
  - The advantage of K-fold CV is that all examples are eventually used for both training and testing
- ◆ True error is estimated as the average

$$E = \frac{1}{K} \sum_{k=1}^K E_k$$

# Leave-one-out Cross-Validation

- ◆ Leave-one-out CV is the extreme case of K-fold CV, where  $K=N$



$$E = \frac{1}{N} \sum_{k=1}^N E_k$$

# How many folds are needed?

- ◆ With a large number of folds
  - (+) The bias of true error estimate is small (i.e., accurate estimate)
  - (−) The variance of true error estimate is large – the K training sets are too similar to one another
  - (−) The computational time will be large (i.e., many experiments)
- ◆ With a small number of folds
  - (+) The computation time is reduced
  - (+) The variance of true error estimate is small
  - (−) The bias of the estimator is large, depending on the learning curve of the classifier
- ◆ In practice, a large dataset often needs a small K, while a very sparse dataset often needs a large K
- ◆ A common choice for K-fold CV is  $K=10$

# Three-way data splits

- ◆ If model selection and true error estimates are to be computed simultaneously, the data needs to be divided into 3 disjoint sets
  - **Training set:** used for learning – to fit the parameters of the classifier
  - **Validation set:** used to tune the parameters of a classifier
  - **Test set:** used only to assess the performance of a fully trained classifier

