

# Support Vector Machines

## 64-360 Algorithmic Learning, part 3

Norman Hendrich

University of Hamburg  
MIN Faculty, Dept. of Informatics  
Vogt-Kölln-Str. 30, D-22527 Hamburg  
hendrich@informatik.uni-hamburg.de

13/06/2012

## Planning and overview: AL part 3

- ▶ 13/06/2012 support vector machines (1)
- ▶ 20/06/2012 support vector machines (2)
- ▶ 27/06/2012 function approximation
- ▶ 04/07/2012 reinforcement learning (1)
- ▶ 11/07/2012 reinforcement learning (2)

# Outline

Introduction

Review of the linear classifier

Maximum margin classification

Soft-margin classification

Kernels and feature maps



# Support Vector Machines

- ▶ a.k.a. *maximum margin classifiers*
- ▶ a family of related
- ▶ supervised
- ▶ learning methods
- ▶ for classification and regression
  
- ▶ try to minimize the classification error
- ▶ while maximizing the geometric margin



# Hype

SVMs are *quite* popular today

- ▶ often the best solutions on classification benchmarks
- ▶ often work without a lot of tuning
- ▶ can handle large data sets
- ▶ an active research area

but

- ▶ good performance is not guaranteed
- ▶ selection of feature maps is critical
- ▶ still requires prior knowledge and experiments
- ▶ and fine-tuning of parameters



## Overall concept and architecture

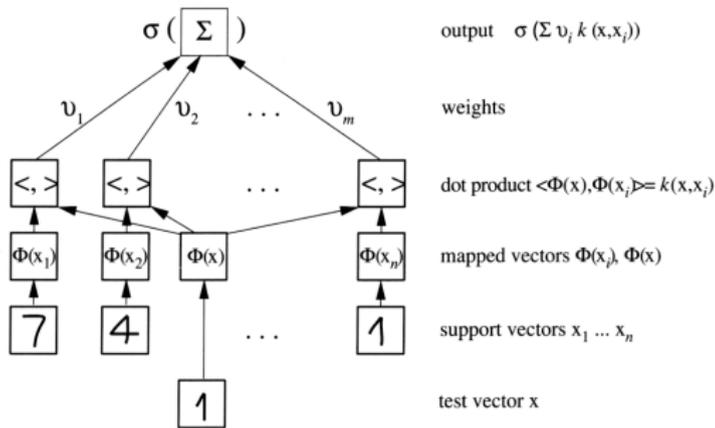
- ▶ select a feature space  $\mathcal{H}$  and a mapping function  $\Phi : x \mapsto \Phi(x)$
- ▶ select a classification (output) function  $\sigma$

$$y(x) = \sigma\left(\sum_i \vartheta_i \langle \Phi(x), \Phi(x_i) \rangle\right)$$

- ▶ during training, find the *support-vectors*  $x_1 \dots x_n$
- ▶ and weights  $\vartheta$  which minimize the classification error
- ▶ map test input  $x$  to  $\Phi(x)$
- ▶ calculate dot-products  $\langle \Phi(x) \Phi(x_i) \rangle$
- ▶ feed linear combination of the dot-products into  $\sigma$
- ▶ get the classification result

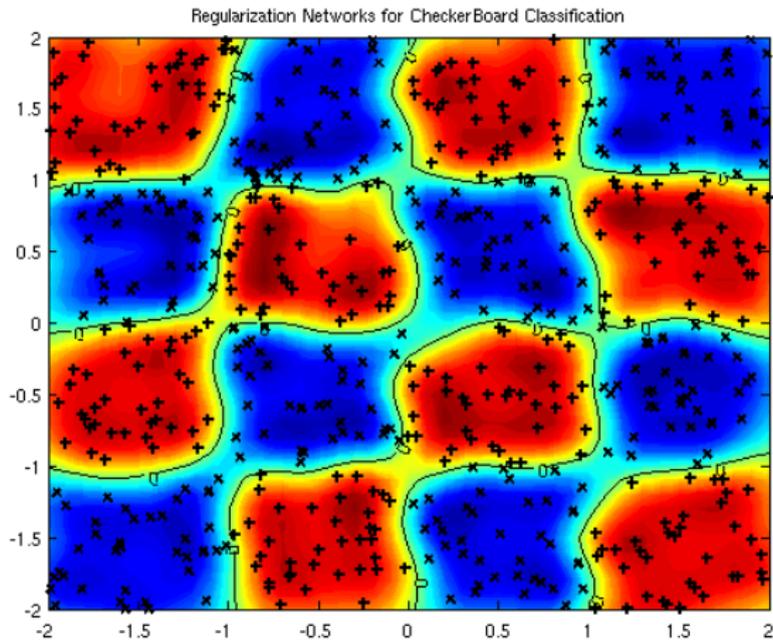
# Block-diagram

Example: handwritten digit recognition (Schölkopf and Smola 2002)



**Figure 1.9** Architecture of SVMs and related kernel methods. The input  $x$  and the expansion patterns (SVs)  $x_i$  (we assume that we are dealing with handwritten digits) are nonlinearly mapped (by  $\Phi$ ) into a feature space  $\mathcal{H}$  where dot products are computed. Through the use of the kernel  $k$ , these two layers are in practice computed in one step. The results are linearly combined using weights  $v_i$ , found by solving a quadratic program (in pattern recognition,  $v_i = y_i \alpha_i$ ; in regression estimation,  $v_i = \alpha_i^+ - \alpha_i^-$ ) or an eigenvalue problem (Kernel PCA). The linear combination is fed into the function  $\sigma$  (in pattern recognition,  $\sigma(x) = \text{sgn}(x + b)$ ; in regression estimation,  $\sigma(x) = x + b$ ; in Kernel PCA,  $\sigma(x) = x$ ).

# Example: learning a checkers board





# History

## Three revolutions in machine learning (Shawe-Taylor & Cristianini 2004)

- ▶ 1960s: efficient algorithms for (linear) pattern detection
  - ▶ e.g., Perceptron (Rosenblatt 1957)
  - ▶ efficient training algorithms
  - ▶ good generalization
  - ▶ but insufficient for nonlinear data
- ▶ 1980s: multi-layer networks and backpropagation
  - ▶ can deal with nonlinear data
  - ▶ but high modeling effort, long training times
  - ▶ and risk of overfitting
- ▶ 1990s: SVMs and related Kernel Methods
  - ▶ “all in one” solution
  - ▶ considerable success on practical applications
  - ▶ based on principled statistical theory



## History: SVM

- ▶ seminal work by Vladimir Vapnik
- ▶ B. E. Boser, I. M. Guyon, and V. N. Vapnik, *A training algorithm for optimal margin classifiers.*, 5th Annual ACM Workshop on COLT, pages 144-152, Pittsburgh, 1992
- ▶ C. Cortes and V. Vapnik, *Support-Vector Networks*, Machine Learning, 20, 1995.  
<http://www.springerlink.com/content/k238jx04hm87j80g/>
- ▶ H. Drucker, C.J.C. Burges, L. Kaufman, A. Smola, and V. Vapnik *Support Vector Regression Machines*, Advances in Neural Information Processing Systems 9, NIPS 1996, 155-161
- ▶ The “bible”: V. Vapnik, *The Nature of Statistical Learning Theory*, Springer, 1995



## References

- ▶ V. Vapnik, *The Nature of Statistical Learning Theory*, Springer, 1995
- ▶ N. Cristianini, J. Shawe-Taylor, *Introduction to Support Vector Machines and other kernel-based learning methods*, Cambridge University Press, 2000
- ▶ J. Shawe-Taylor, N. Cristianini, *Kernel Methods for Pattern Analysis*, Cambridge University Press, 2004
- ▶ B. Schölkopf, A. J. Smola, *Learning with Kernels*, MIT Press, 2002
- ▶ L. Bottou, O. Chapelle, D. DeCoste, J. Weste (Eds), *Large-Scale Kernel Machines*, MIT Press, 2007

## References: web resources

- ▶ [www.kernel-machines.org/](http://www.kernel-machines.org/)
- ▶ A. W. Moore, *Support Vector Machines*, [www.cs.cmu.edu/~awm](http://www.cs.cmu.edu/~awm), 2003
- ▶ S. Bloehdorn, *Maschinelles Lernen*, <http://www.aifb.uni-karlsruhe.de/WBS/pci/ML/SVMs.pdf>
- ▶ C.-C. Chang & C.-J. Lin, *libsvm*, <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>
- ▶ W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes – The Art of Scientific Computing*, Cambridge University Press, 2007 (all algorithms on CD-ROM)
- ▶ S. R. Gunn, *Support Vector Classification and Regression*, Technical Report, University of Southampton, 1998

## Review: binary classification

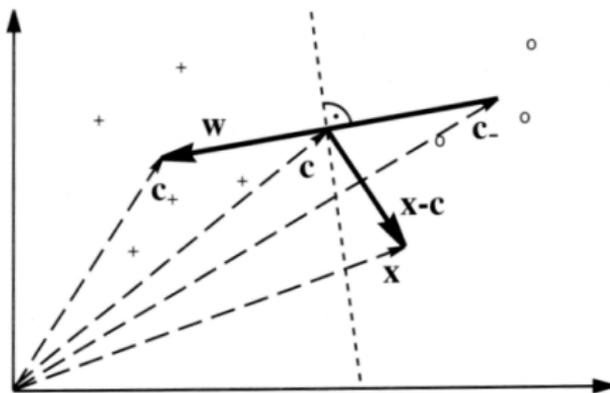
task:

- ▶ classify input test patterns  $x$
- ▶ based on previously learned training patterns
- ▶ simplest case is binary classification,
- ▶ two-classes  $y(x) = \{+1, -1\}$

A first example algorithm:

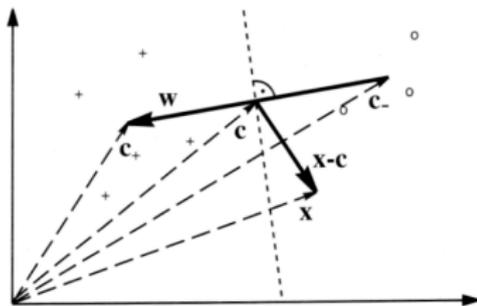
- ▶ classify based on distance to the
- ▶ center-of-mass of the training pattern clusters
- ▶ result can be written as  $y = \text{sgn}(\sum_i w_i \cdot x_i + b)$

# Simple classification example



**Figure 1.1** A simple geometric classification algorithm: given two classes of points (depicted by 'o' and '+'), compute their means  $c_+$ ,  $c_-$  and assign a test pattern  $x$  to the one whose mean is closer. This can be done by looking at the dot product between  $x - c$  (where  $c = (c_+ + c_-)/2$ ) and  $w := c_+ - c_-$ , which changes sign as the enclosed angle passes through  $\pi/2$ . Note that the corresponding decision boundary is a hyperplane (the dotted line) orthogonal to  $w$ .

## Simple classification example (cont'd)



- ▶ two classes of data points ('o' and '+')
- ▶ calculate the means of each cluster (center of mass)
- ▶ assign test pattern  $x$  to the nearest cluster
- ▶ can be written as  $y = \text{sgn}(\sum_{i=1}^m \alpha_i \langle x, x_i \rangle + b)$
- ▶ with constant weights  $\alpha_j = \{\frac{1}{m_+}, \frac{1}{m_-}\}$

## Simple classification example (cont'd)

- ▶ centers of mass:

$$c_+ = \frac{1}{m_+} \sum_{\{i|y_i=+1\}} x_i,$$

$$c_- = \frac{1}{m_-} \sum_{\{i|y_i=-1\}} x_i,$$

- ▶ boundary point  $c$ :  $c = (c_+ + c_-)/2$

- ▶ classification:  $y = \text{sgn}(\langle (x - c), w \rangle)$

- ▶ norm:  $\|x\| := \sqrt{\langle x, x \rangle}$

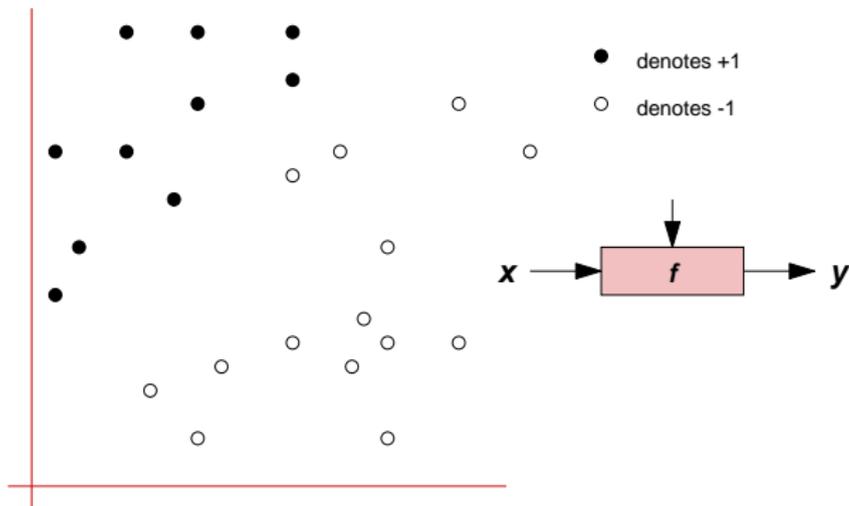
- ▶ rewrite:  $y = \text{sgn}(\langle (x, c_+) \rangle - \langle (x, c_-) \rangle + b)$

$$\text{with } b = (\|c_-\|^2 - \|c_+\|^2)/2$$

- ▶ all together:

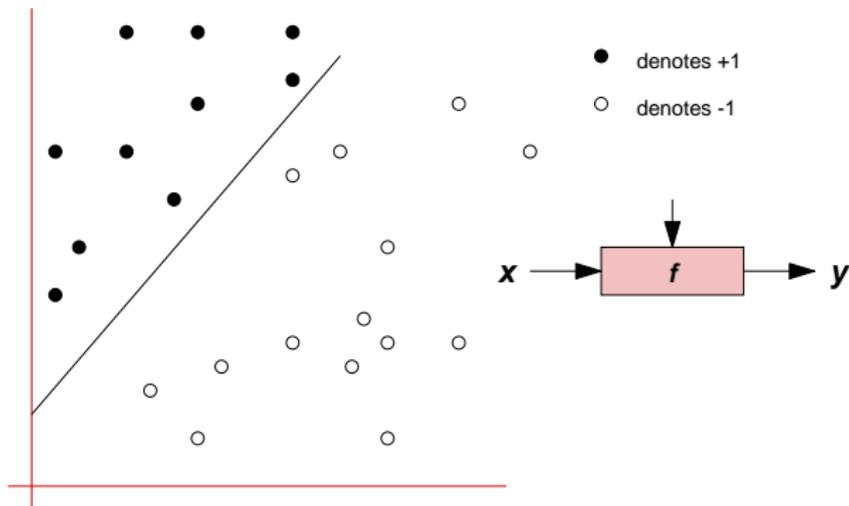
$$y = \text{sgn}\left(\frac{1}{m_+} \sum_{\{i|y_i=+1\}} x_i \langle x, x_i \rangle - \frac{1}{m_-} \sum_{\{i|y_i=-1\}} x_i \langle x, x_i \rangle + b\right)$$

# Linear classification



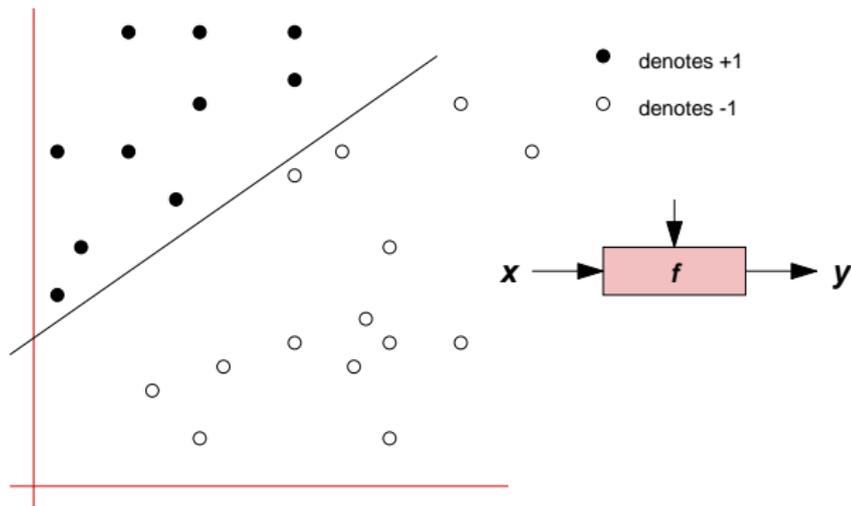
- find  $w$  and  $b$ , so that  $y(x, w, b) = \text{sgn}(w \cdot x - b)$

# Linear classification



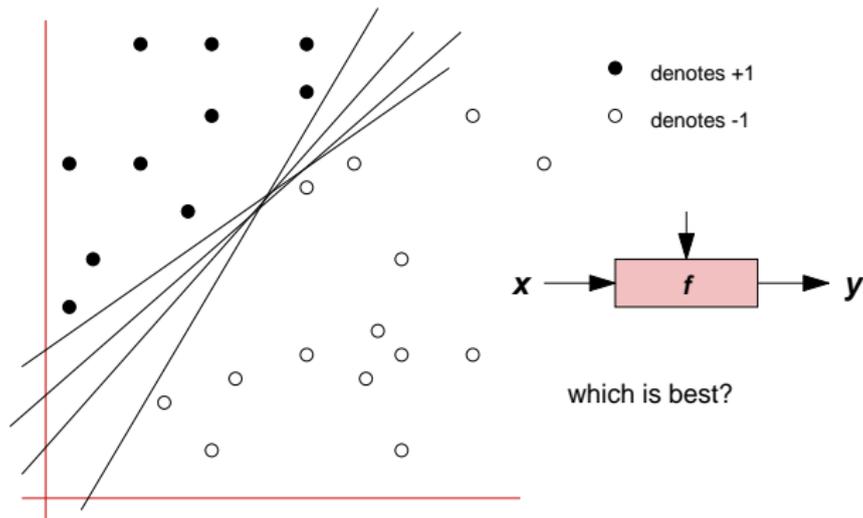
- ▶ one possible decision boundary

# Linear classification



► and another one

# Linear classification



► which boundary is best?



## Remember: Perceptron

- ▶ can use the Perceptron learning algorithm
- ▶ to find a valid decision boundary
  
- ▶ convergence is guaranteed,
- ▶ iff the data is separable
  
- ▶ algorithm stops as soon as a solution is found
- ▶ but we don't know which boundary will be chosen



# Perceptron training algorithm

Input: training data  $S = \{x_i, y_i\}$

Initialize:  $w \leftarrow 0, b \leftarrow 0$

Repeat:

$err \leftarrow 0$

    for  $i = 1, \dots, l$  do

        compute  $f(x_i) = \text{sgn}(\langle w, \Phi(x_i) \rangle + b)$

        if  $f(x_i) \neq y_i$  then

$w \leftarrow w + y_i \Phi(x_i)$

$b \leftarrow b + y_i$

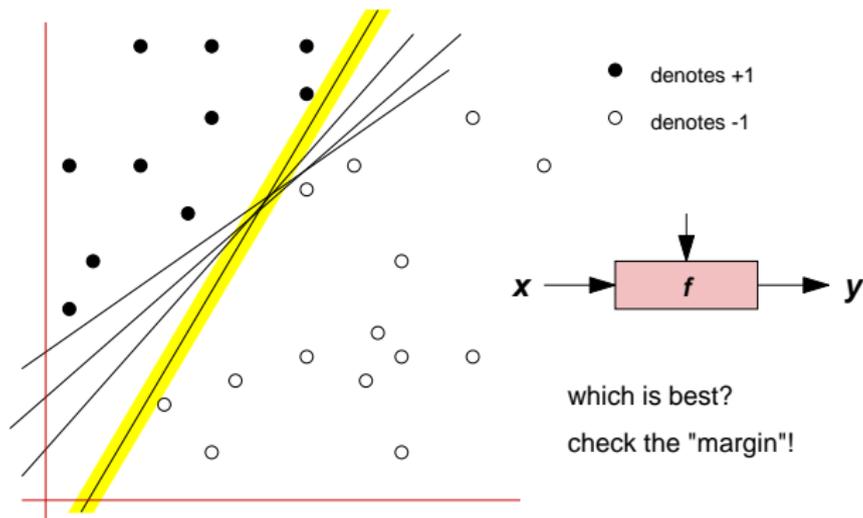
        end if

    end for

until  $err = 0$

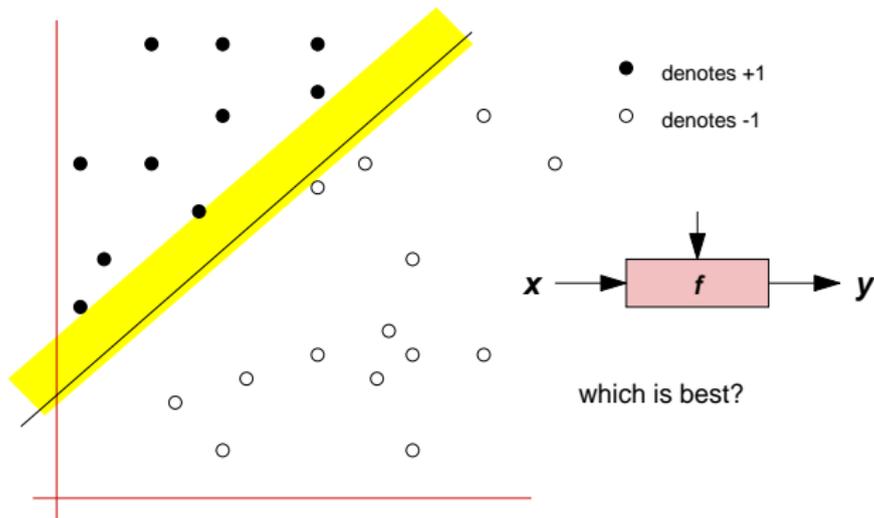
return  $w, b$

# The classifier margin



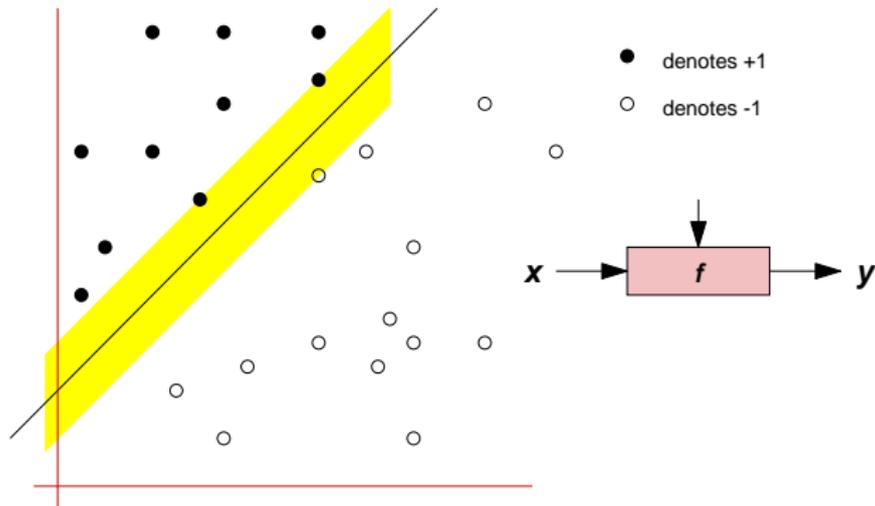
- ▶ define the *margin* as the width that the boundary could be increased before hitting a data point.

# The classifier margin



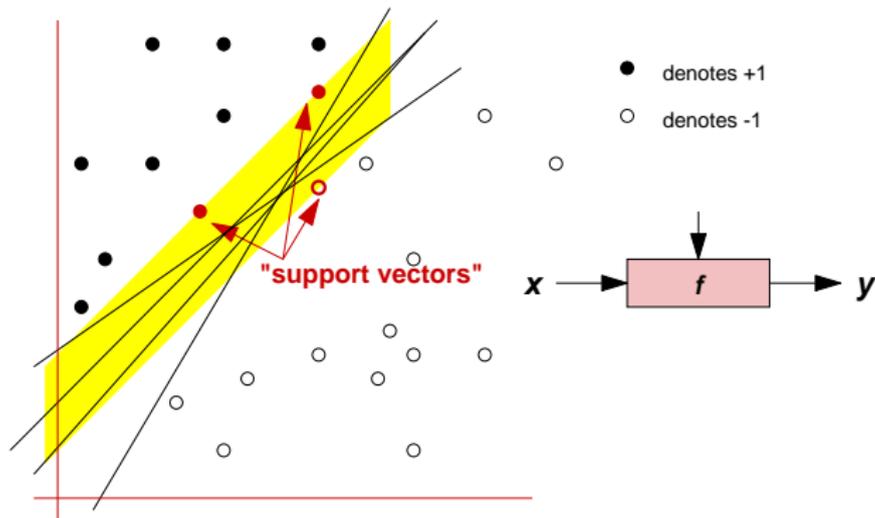
- ▶ a second example: margin not symmetrical

# Maximum margin classifier



- ▶ the classifier with the largest margin
- ▶ the simplest kind of SVM (called the linear SVM)

# Support vectors



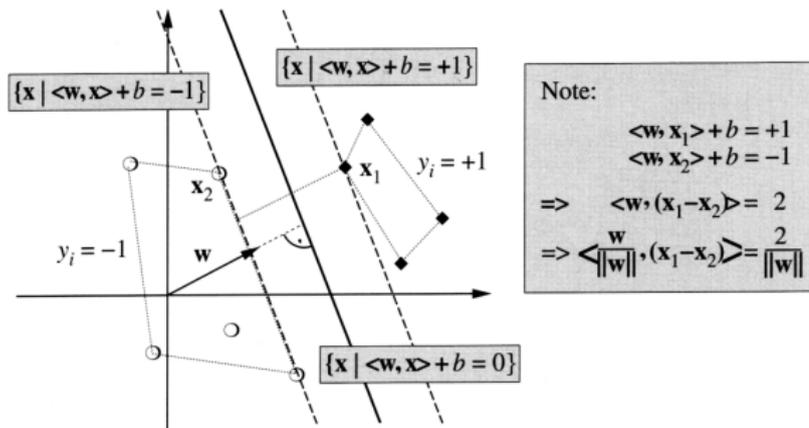
- ▶ data points that limit the margin are called the *support vectors*



## Why maximum margin?

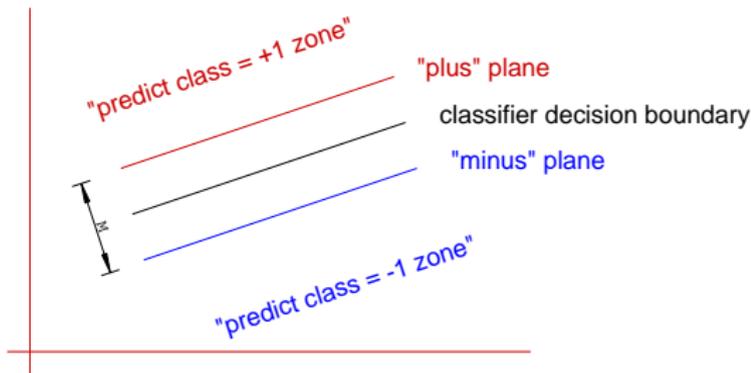
- ▶ intuitively, feels safest
- ▶ least chance of misclassification if the decision boundary is not exactly correct
- ▶ statistical theory (“VC dimension”) indicates that maximum margin is good
- ▶ empirically, works very well
  
- ▶ note: far fewer support-vectors than data points (unless overfitted)
- ▶ note: the model is immune against removal of all non-support-vector data points

# The geometric interpretation



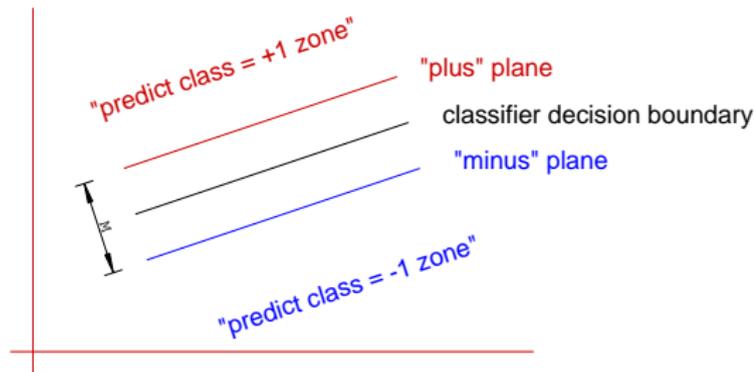
**Figure 1.5** A binary classification toy problem: separate balls from diamonds. The *optimal hyperplane* (1.23) is shown as a solid line. The problem being separable, there exists a weight vector  $\mathbf{w}$  and a threshold  $b$  such that  $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) > 0$  ( $i = 1, \dots, m$ ). Rescaling  $\mathbf{w}$  and  $b$  such that the point(s) closest to the hyperplane satisfy  $|\langle \mathbf{w}, \mathbf{x}_i \rangle + b| = 1$ , we obtain a *canonical form*  $(\mathbf{w}, b)$  of the hyperplane, satisfying  $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$ . Note that in this case, the *margin* (the distance of the closest point to the hyperplane) equals  $1/\|\mathbf{w}\|$ . This can be seen by considering two points  $\mathbf{x}_1, \mathbf{x}_2$  on opposite sides of the margin, that is,  $\langle \mathbf{w}, \mathbf{x}_1 \rangle + b = 1, \langle \mathbf{w}, \mathbf{x}_2 \rangle + b = -1$ , and projecting them onto the hyperplane normal vector  $\mathbf{w}/\|\mathbf{w}\|$ .

# Step by step: calculating the margin width



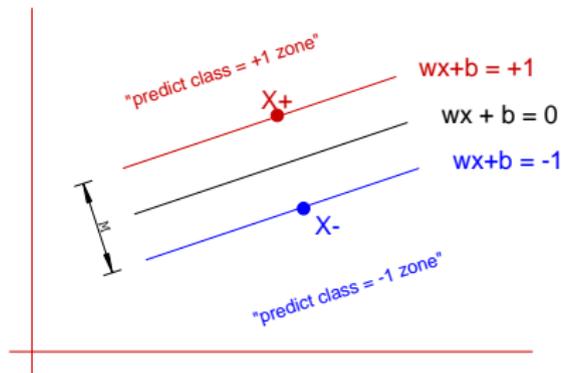
- ▶ how to represent the boundary (hyperplane)
- ▶ and the margin width  $M$
- ▶ in  $m$  input dimensions?

# Calculating the margin width



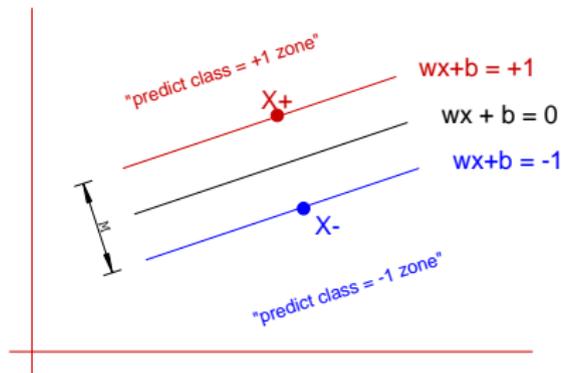
- ▶ plus-plane:  $\{x : w \cdot x + b = +1\}$
- ▶ minus-plane:  $\{x : w \cdot x + b = -1\}$
- ▶ classify pattern as  $+1$  if  $w \cdot x + b \geq +1$   
 and  $-1$  if  $w \cdot x + b \leq -1$

# Calculating the margin width



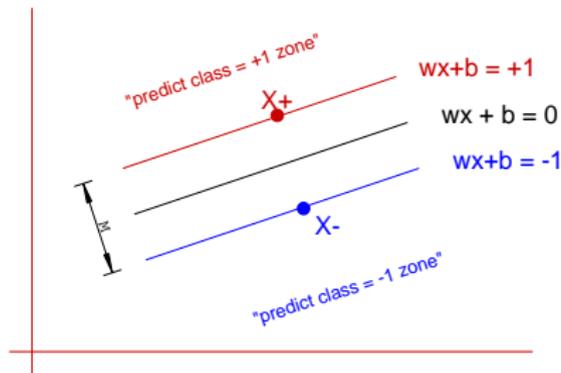
- ▶  $w$  is perpendicular to the decision boundary
- ▶ and the plus-plane and minus-plane
- ▶ proof: consider two points  $u$  and  $v$  on the plus-plane and calculate  $w \cdot (u - v)$

# Calculating the margin width



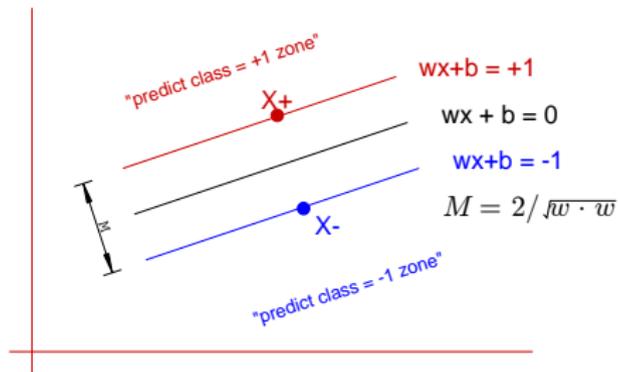
- ▶ select point  $X^+$  on the plus plane
- ▶ and nearest point  $X^-$  on the minus plane
- ▶ of course, margin width  $M = |X^+ - X^-|$
- ▶ and  $X^+ = X^- + \lambda w$  for some  $\lambda$

# Calculating the margin width



- ▶  $w \cdot (X^- + \lambda w) + b = 1$
- ▶  $w \cdot X^- + b + \lambda w \cdot w = 1$
- ▶  $-1 + \lambda w \cdot w = 1$
- ▶  $\lambda = \frac{2}{w \cdot w}$

# Calculating the margin width



- ▶  $\lambda = \frac{2}{w \cdot w}$
- ▶  $M = |X^+ - X^-| = |\lambda w| = \lambda |w|$
- ▶  $M = \lambda \sqrt{w \cdot w} = 2 / \sqrt{w \cdot w}$

# Training the maximum margin classifier

Given a guess of  $w$  and  $b$  we can

- ▶ compute whether all data points are in the correct half-planes
- ▶ compute the width of the margin

So: write a program to search the space of  $w$  and  $b$  to find the widest margin that still correctly classifies all training data points.

- ▶ but how?
- ▶ gradient descent? simulated annealing? ...
- ▶ usually, Quadratic programming

# Learning via Quadratic Programming

- ▶ QP is a well-studied class of optimization algorithms
- ▶ maximize a quadratic function of real-valued variables
- ▶ subject to linear constraints
  
- ▶ could use standard QP program libraries
- ▶ e.g. MINOS  
[http://www.sbsi-sol-optimize.com/asp/sol\\_products\\_minos.htm](http://www.sbsi-sol-optimize.com/asp/sol_products_minos.htm)
- ▶ e.g. LOQO <http://www.princeton.edu/~rvdb/loqo>
  
- ▶ or algorithms streamlined for SVM (e.g. large data sets)

# Quadratic Programming

General problem:

▶ find  $\arg \max_u (c + d^T u + \frac{1}{2} u^T R u)$

▶ subject to  $n$  linear inequality constraints

$$a_{11}u_1 + a_{12}u_2 + \dots + a_{1m}u_m \leq b_1$$

$$a_{21}u_1 + a_{22}u_2 + \dots + a_{2m}u_m \leq b_2$$

...

$$a_{n1}u_1 + a_{n2}u_2 + \dots + a_{nm}u_m \leq b_n$$

▶ subject to  $e$  additional linear equality constraints

$$a_{(n+1)1}u_1 + a_{(n+1)2}u_2 + \dots + a_{(n+1)m}u_m = b_{n+1}$$

...

$$a_{(n+e)1}u_1 + a_{(n+e)2}u_2 + \dots + a_{(n+e)m}u_m = b_{n+1}$$

## QP for the maximum margin classifier

Setup of the Quadratic Programming for SVM:

- ▶  $M = \lambda \sqrt{w \cdot w} = 2 / \sqrt{w \cdot w}$
- ▶ for largest  $M$ , we want to minimize  $w \cdot w$
- ▶ assuming  $R$  data points  $(x_k, y_k)$  with  $y_k = \pm 1$
- ▶ there are  $R$  constraints:
  - $w \cdot x_k + b \geq +1$  if  $y_k = +1$
  - $w \cdot x_k + b \leq -1$  if  $y_k = -1$

## QP for the maximum margin classifier

- ▶ solution of the QP problem is possible
- ▶ but difficult, because of the complex constraints

Instead, switch to the *dual representation*

- ▶ use the “Lagrange multiplier” trick
- ▶ introduce new dummy variables  $\alpha_i$
- ▶ this allows to rewrite with simple inequalities  $\alpha_i \geq 0$
- ▶ solve the optimization problem, find  $\alpha_i$
- ▶ from the  $\alpha_i$ , find the separating hyperplane ( $w$ )
- ▶ from the hyperplane, find  $b$



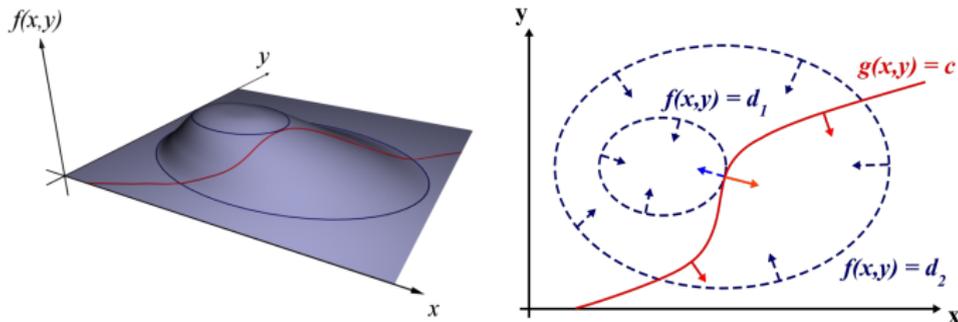
## Reminder: Lagrange multiplier

- ▶ maximize  $f(x, y)$  subject to  $g(x, y) = c$
- ▶ introduce a new goal function

$$\Lambda(x, y, \lambda) = f(x, y) + \lambda \cdot (g(x, y) - c)$$

- ▶ new variable  $\lambda$ , the Lagrange multiplier
- ▶ if  $f(x_0, y_0)$  is maximum of  $f(x, y)$  for the original problem, then there exists  $\lambda_0$  such that  $(x_0, y_0, \lambda_0)$  is a stationary point for the Lagrange function  $\Lambda$ , i.e. its partial derivatives are zero
- ▶ solve  $\nabla_{x,y,\lambda} \Lambda(x, y, \lambda) = \left( \frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial \lambda} \right) \Lambda = 0$
- ▶ note:  $\nabla_{\lambda} \Lambda(x, y, \lambda) = 0$  implies  $g(x, y) = c$
- ▶ handling inequalities  $g(x, y) \geq c$  is slightly more complex, called the Karush-Kuhn-Tucker (KKT) method

## Lagrange multiplier: geometric interpretation



- ▶ maximize  $f(x, y)$  subject to  $g(x, y) = c$  (red line)
- ▶ contour line of  $f$  must touch  $g$  tangentially
- ▶ otherwise, moving along  $g$  would increase/decrease  $f$
- ▶ this also implies gradients must be (anti-)parallel, factor  $\lambda$
- ▶  $\nabla_{x,y} f = \lambda \nabla_{x,y} g$  and  $g(x, y) = c$

## The dual optimization problem

- ▶ introduce Lagrange multipliers  $\alpha_i$
- ▶ optimize new goal function

$$\max \mathcal{D}(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n y_i \alpha_i y_j \alpha_j \Phi(x_i)^T \Phi(x_j)$$

subject to  $\alpha_i \geq 0 \quad \forall i$  and  $\sum_i y_i \alpha_i = 0$

- ▶ optimal separating hyperplane  $w^*$  is then calculated from the solution  $\alpha^*$  of the above problem,

$$w^* = \sum_i \alpha_i^* y_i \Phi(x_i)$$

- ▶ finally, solve one support-vector constraint to get the bias  $b^*$

## Summary: Linear SVM

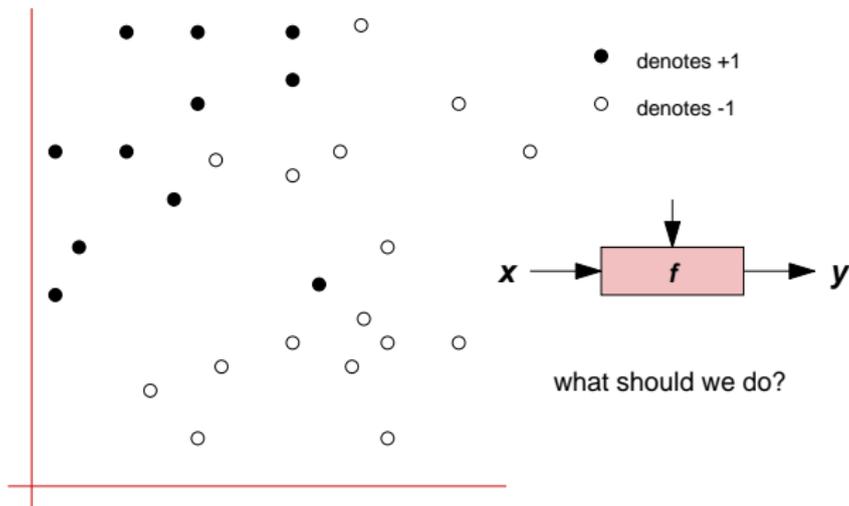
- ▶ based on the classical linear classifier
- ▶ maximum margin concept
- ▶ limiting data points are called Support Vectors
  
- ▶ solution via Quadratic Programming
- ▶ dual formulation (usually) easier to solve



## Classification of noisy input data?

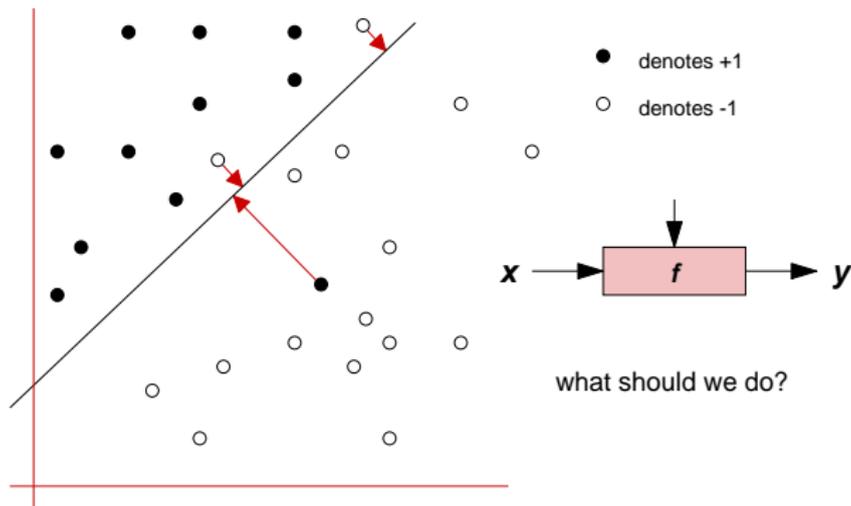
- ▶ actual “real world” training data contains noise
- ▶ usually, several “outlier” patterns
- ▶ for example, mis-classified training data
  
- ▶ at least, reduced error-margins
- ▶ or worse, training set not linearly separable
- ▶ complicated decision boundaries
  
- ▶ complex kernels can handle this (see below)
- ▶ but not always the best idea
- ▶ risk of overfitting
  
- ▶ instead, allow some patterns to violate the margin constraints

# The example data set, modified



- ▶ not linearly separable!
- ▶ trust every data point?

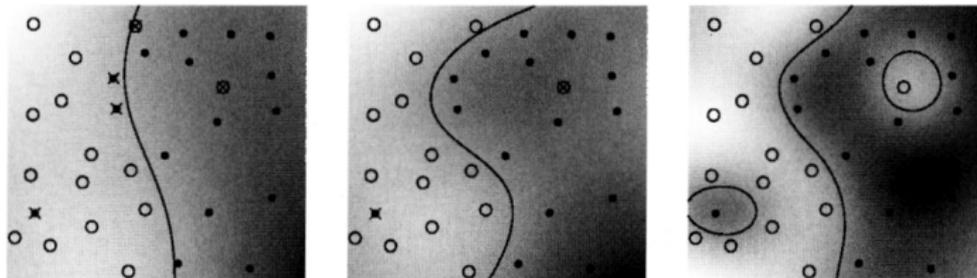
## Example data set, and one example classifier



- ▶ three points misclassified
- ▶ two with small margin, one with large margin

# Noisy input data? Another toy example

LWK, page 10



**Figure 1.2** 2D toy example of binary classification, solved using three models (the decision boundaries are shown). The models vary in complexity, ranging from a simple one (*left*), which misclassifies a large number of points, to a complex one (*right*), which “trusts” each point and comes up with solution that is consistent with all training points (but may not work well on new points). As an aside: the plots were generated using the so-called soft-margin SVM to be explained in Chapter 7; cf. also Figure 7.10.

- ▶ allow errors?
- ▶ trust every data point?

# Soft-margin classification

Cortes and Vapnik, 1995

- ▶ allow some patterns to violate the margin constraints
- ▶ find a compromise between large margins
- ▶ and the number of violations

Idea:

- ▶ introduce slack-variables  $\xi = (\xi_1 \dots \xi_n)$ ,  $\xi_i \geq 0$
- ▶ which measure the margin violation (or classification error) on pattern  $x_i$ :  $y(x_i)(w \cdot \Phi(x_i) + b) \geq 1 - \xi_i$
- ▶ introduce one global parameter  $C$  which controls the compromise between large margins and the number of violations

## Soft-margin classification

- ▶ introduce slack-variables  $\xi_i$
- ▶ and global control parameter  $C$

$$\max_{w, b, \xi} \mathcal{P}(w, b, \xi) = \frac{1}{2} w^2 + C \sum_{i=1}^n \xi_i$$

subject to:

$$\forall i: y(x_i)(w \cdot \Phi(x_i) + b) \geq 1 - \xi_i$$

$$\forall i: \xi_i \geq 0$$

- ▶ problem is now very similar to the hard-margin case
- ▶ again, the dual representation is often easier to solve

## Lagrange formulation of the soft-margin SVM

- ▶ again, introduce Lagrange multipliers  $\alpha_i \geq 0$

$$\mathcal{L}(w, b, \xi, \alpha) = \frac{1}{2}w^2 + C \sum_{i=1}^n \xi_i - \sum_{i=1}^n \alpha_i (y_i(w^T \Phi(x_i) + b) - 1 + \xi_i)$$

- ▶ minimize this function

$$\min_{w, b, \xi} \mathcal{L}(w, b, \xi, \alpha) \quad \text{subject to} \quad \forall_i \quad \xi_i \geq 0$$

- ▶ weight vector and bias are then calculated from the solution  $\alpha_i$  as above



## How to select the control parameter?

- ▶ of course, the optimization result depends on the specified control parameter  $C$
- ▶ how to select the value of  $C$ ?
  
- ▶ depends on the application and training data
- ▶ Numerical Recipes recommends the following:
  - ▶ start with  $C = 1$
  - ▶ then try to increase or decrease by powers of 10
  - ▶ until you find a broad plateau where the exact value of  $C$  doesn't matter much
  - ▶ good SVM solution should classify most patterns correctly,
  - ▶ with many  $\alpha_i = 0$  and many  $\alpha_i = C$ , but only a few in between



## Summary: soft-margin SVM

- ▶ same concept as the linear SVM
- ▶ try to maximize the decision margin
- ▶ allow some patterns to violate the margin constraints
- ▶ compromise between large margin and number of violations
  
- ▶ introduce a control parameter  $C$
- ▶ and new inequality parameters  $\xi_i$  (slack)
- ▶ again, can be written as a QP problem
- ▶ again, dual formulation easier to solve

## Nonlinearity through feature maps

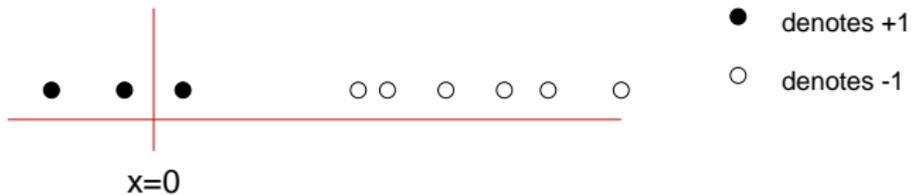
General idea:

- ▶ introduce a function  $\Phi$  which maps the input data into a higher dimensional *feature space*

$$\Phi : x \in X \mapsto \Phi(x) \in \mathcal{H}$$

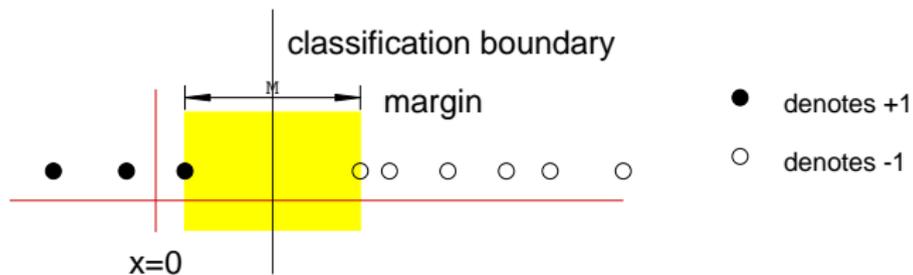
- ▶ similar to hidden layers of multi-layer ANNs
- ▶ explicit mappings can be expensive in terms of CPU and/or memory (especially in high dimensions)
- ▶ “*Kernel functions*” achieve this mapping *implicitly*
- ▶ often, very good performance

## Example 1-dimensional data set



- what would the linear SVM do with these patterns?

## Example 1-dimensional data set



- ▶ what would the linear SVM do with these patterns?
- ▶ not a big surprise!
- ▶ maximum margin solution

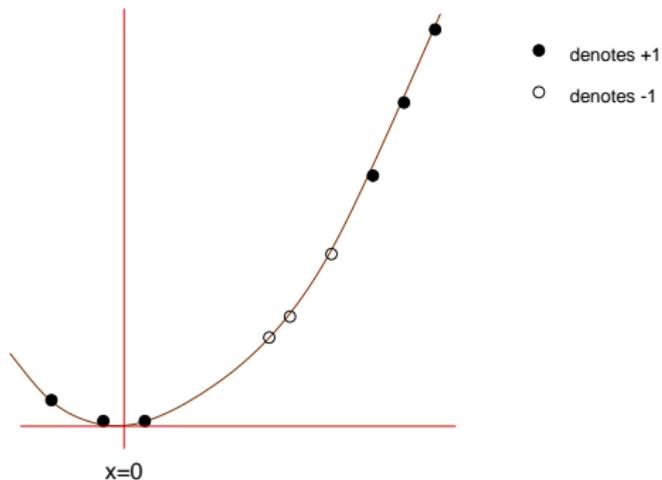
## Harder 1-dimensional data set



- denotes +1
- denotes -1

- ▶ and now?
- ▶ doesn't look like "outliers"
- ▶ so, soft-margin won't help a lot

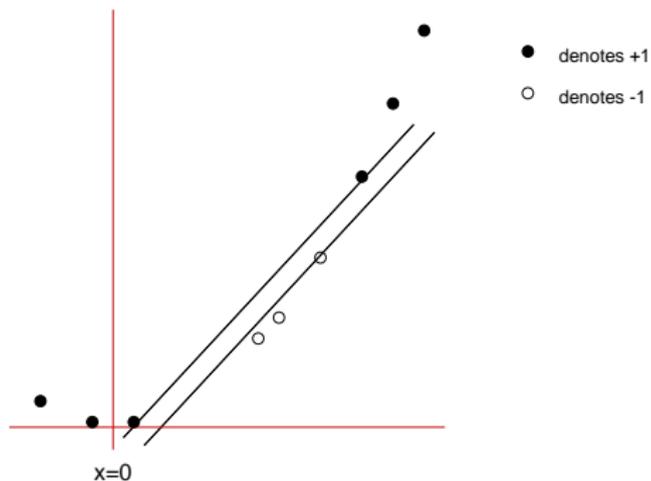
## Harder 1-dimensional data set



- ▶ permit non-linear basis functions
- ▶  $z_k = (x_k, x_k^2)$



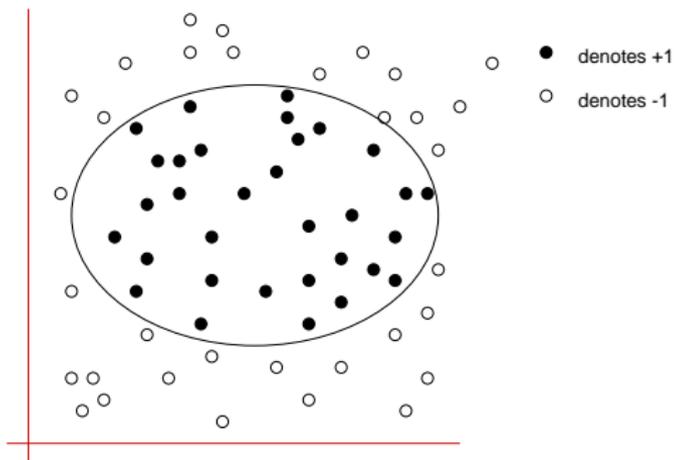
## Harder 1-dimensional data set



▶  $z_k = (x_k, x_k^2)$

▶ data is now linearly separable!

## Similar for 2-dimensional data set



- ▶ clearly not linearly separable in 2D
- ▶ introduce  $z_k = (x_k, y_k, \sqrt{2}x_k y_k)$

# Common SVM feature maps

## basis functions

- ▶  $z_k = (\text{polynomial terms of } x_k \text{ of degree 1 to } q)$
- ▶  $z_k = (\text{radial basis functions of } x_k)$
- ▶  $z_k = (\text{sigmoid functions of } x_k)$
- ▶ ...
- ▶ combinations of the above

Note:

- ▶ feature map  $\Phi$  only used in inner products
- ▶ for training, information on pairwise inner products is sufficient



## Kernel: definition

Definition 1 (Kernel): A *Kernel* is a function  $K$ , such that for all  $x, z \in \mathcal{X}$ :

$$K(x, z) = \langle \phi(x), \phi(z) \rangle.$$

where  $\Phi$  is a mapping from  $\mathcal{X}$  to an (inner product) feature space  $\mathcal{F}$ .



## Example: polynomial Kernel

- ▶ consider the mapping:

$$\Phi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \in \mathbb{R}^3$$

- ▶ evaluation of dot products:

$$\begin{aligned} \langle \Phi(x), \Phi(z) \rangle &= \langle (x_1^2, \sqrt{2}x_1x_2, x_2^2), (z_1^2, \sqrt{2}z_1z_2, z_2^2) \rangle \\ &= x_1^2z_1^2 + 2x_1x_2z_1z_2 + x_2^2z_2^2 \\ &= (x_1z_1 + x_2z_2)^2 = \langle x, z \rangle^2 = \kappa(x, z) \end{aligned}$$

- ▶ kernel does not uniquely determine the feature space:

$$\Phi'(x) = (x_1^2, x_2^2, x_1x_2, x_2x_1) \in \mathbb{R}^4$$

$$\text{also fits to } k(x, z) = \langle x, z \rangle^2$$

## Example: quadratic kernel, $m$ dimensions

- ▶  $x = (x_1, \dots, x_m)$
- ▶  $\Phi(x) = ($ 

$$\begin{aligned}
 &1, \\
 &\sqrt{2}x_1, \sqrt{2}x_2, \dots, \sqrt{2}x_m, \\
 &x_1^2, x_2^2, \dots, x_m^2, \\
 &\sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \dots, \sqrt{2}x_{m-1}x_m )
 \end{aligned}$$
- ▶ constant, linear, pure quadratic, cross quadratic terms
- ▶ in total  $(m+2)(m+1)/2$  terms (roughly  $m^2/2$ )
- ▶ so, complexity of evaluating  $\Phi(x)$  is  $O(m^2)$
- ▶ for example,  $m = 100$  implies 5000 terms...

## Example: quadratic kernel, scalar product

$$\Phi(x) \cdot \Phi(y) = \begin{pmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \dots \\ x_1^2 \\ x_2^2 \\ \dots \\ \sqrt{2}x_1x_2 \\ \sqrt{2}x_1x_3 \\ \dots \\ \sqrt{2}x_{m-1}x_m \end{pmatrix} \cdot \begin{pmatrix} 1 \\ \sqrt{2}y_1 \\ \sqrt{2}y_2 \\ \dots \\ y_1^2 \\ y_2^2 \\ \dots \\ \sqrt{2}y_1y_2 \\ \sqrt{2}y_1y_3 \\ \dots \\ \sqrt{2}y_{m-1}y_m \end{pmatrix} =$$

$$1 + \sum_{i=1}^m 2x_iy_i + \sum_{i=1}^m x_i^2y_i^2 + \sum_{i=1}^m \sum_{j=1}^m 2x_ix_jy_iy_j$$

## Example: scalar product

- ▶ calculating  $\langle \Phi(x), \Phi(y) \rangle$  is  $O(m^2)$
- ▶ for comparison, calculate  $(x \cdot y + 1)^2$  :
  
- ▶  $(x \cdot y + 1)^2 = ((\sum_{i=1}^m x_i \cdot y_i) + 1)^2$ 

$$= (\sum_{i=1}^m x_i y_i)^2 + 2(\sum_{i=1}^m x_i y_i) + 1$$

$$= \sum_{i=1}^m \sum_{j=1}^m x_i y_i x_j y_j + 2 \sum_{i=1}^m x_i y_i + 1$$

$$= \sum_{i=1}^m (x_i y_i)^2 + 2 \sum_{i=1}^m \sum_{j=1}^m x_i y_i x_j y_j + 2 \sum_{i=1}^m x_i y_i + 1$$

$$= \Phi(x) \cdot \Phi(y)$$
  
- ▶ we can replace  $\langle \Phi(x), \Phi(y) \rangle$  with  $(x \cdot y + 1)^2$ , which is  $O(m)$

## Polynomial kernels

- ▶ the learning algorithm only needs  $\langle \Phi(x), \Phi(y) \rangle$
- ▶ for the quadratic polynomial, we can replace this by  $(\langle x, y \rangle + 1)^2$
- ▶ optional, use scale factors:  $(a\langle x, y \rangle + b)^2$
- ▶ calculating one scalar product drops from  $O(m^2)$  to  $O(m)$
- ▶ overall training algorithm then is  $O(mR^2)$
  
- ▶ same trick also works for cubic and higher degree
- ▶ cubic polynomial kernel:  $(a\langle x, y \rangle + b)^3$ , includes all  $m^3/6$  terms up to degree 3
- ▶ quartic polynomial kernel:  $(a\langle x, y \rangle + b)^4$  includes all  $m^4/24$  terms up to degree 4
- ▶ etc.

## Polynomial kernels

- ▶ for polynomial kernel of degree  $d$ , we use  $(\langle x, y \rangle + 1)^d$
- ▶ calculating the scalar product drops from  $O(m^d)$  to  $O(m)$
  
- ▶ algorithm implicitly uses an enormous number of terms
- ▶ high theoretical risk of overfitting
- ▶ but often works well in practice
  
- ▶ note: same trick is used to evaluate a test input:  

$$y(x_t) = \sum_{i=1}^R \alpha_k y_k (\langle x_k, x \rangle + 1)^d$$
- ▶ note:  $\alpha_k = 0$  for non-support vectors, so overall  $O(mS)$  with the number of support vectors  $S$ .

## Kernel “design”

How to get up a useful kernel function?

- ▶ derive it directly from explicit feature mappings
- ▶ design a similarity function for your input data, then check whether it is a valid kernel function
- ▶ use the application domain to guess useful values of any kernel parameters (scale factors)
- ▶ for example, for polynomial kernels try to keep  $(a\langle x, y \rangle + b)$  within the range  $\pm 1$  for all  $i$  and  $j$

## Kernel composition

Given Kernels  $K_1$  and  $K_2$  over  $X \times X$ , the following functions are also kernels:

- ▶  $K(x, z) = \alpha K_1(x, z), \alpha \in \mathbb{R}^+$ ;
- ▶  $K(x, z) = K_1(x, z) + c, c \in \mathbb{R}^+$ ;
- ▶  $K(x, z) = K_1(x, z) + K_2(x, z)$ ;
- ▶  $K(x, z) = K_1(x, z) \cdot K_2(x, z)$ ;
- ▶  $K(x, z) = x' B z, X \subseteq \mathbb{R}^n, B$  pos. sem.-def.



# Gaussian Kernel

a.k.a radial basis function kernel

$$K(x, z) = \exp\left(-\frac{\|x - z\|^2}{2\sigma^2}\right)$$

- ▶ with “bandwidth” parameter  $\sigma$
- ▶ kernel evaluation depends on distance of  $x$  and  $z$
- ▶ *local neighborhood* classification
- ▶ initialize  $\sigma$  to a characteristic distance between nearby patterns in feature space
- ▶ large distance implies orthogonal patterns



## Gaussian Kernel: need to scale your data

- ▶ Gaussian kernel has only one parameter
- ▶ bandwidth  $\sigma$  needs to match **all** dimensions of your data
- ▶ problematic if different dimensions of the input data have largely different numerical values
- ▶ bigger numbers would dominate the scalar product calculations (e.g. size in meters vs. distance in light-years vs. weight in micrograms, etc.)
- ▶ pre-scale all data to consistent scale before training the SVM
- ▶ afterwards, adjust  $\sigma$  and soft-margin parameter  $C$
- ▶ search  $(\sigma, C)$  parameter space for best classification

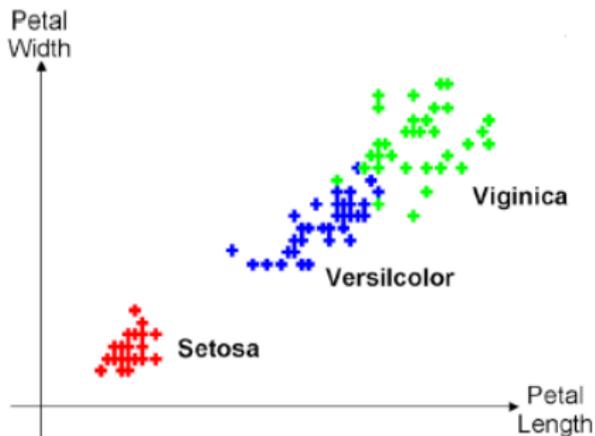
## Mahalanobis distance

- ▶ adjust scale of given data based on the variance of the data
- ▶ input vectors  $x = (x_1, x_2, x_3, \dots, x_n)^T$  from set  $X$
- ▶ with mean  $\mu = (\mu_1, \mu_2, \mu_3, \dots, \mu_n)^T$
- ▶ and covariance matrix  $S$ :

$$D_M(x) = \sqrt{(x - \mu)^T S^{-1} (x - \mu)}.$$

- ▶ note: Euclidean distance is special case for this, iff the covariance matrix of the dataset  $X$  is the identity matrix
- ▶ the “official” way to auto-scale data

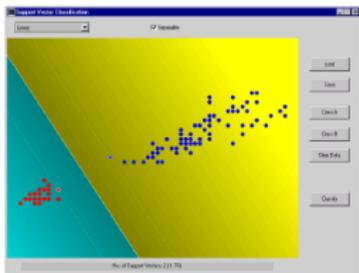
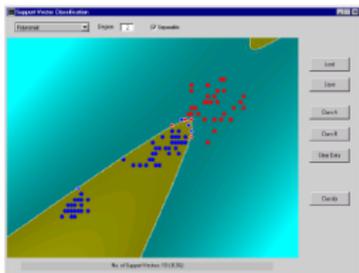
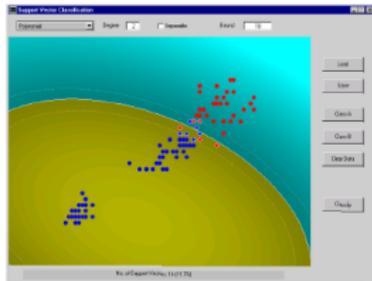
## Kernel functions example: Iris dataset



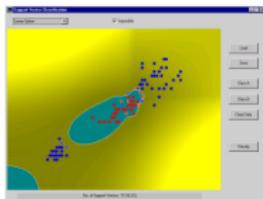
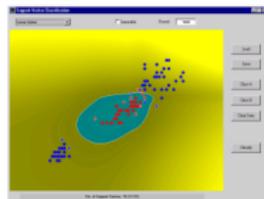
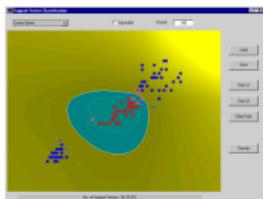
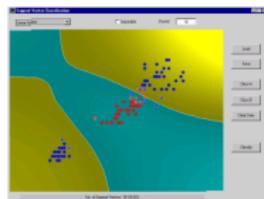
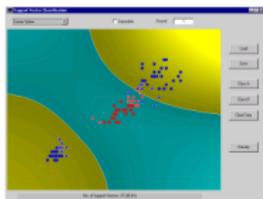
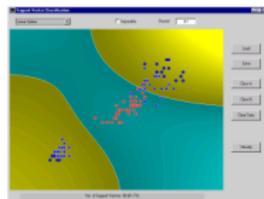
- ▶ three classes of iris flowers
- ▶ classified by petal width and length

(this and the following pictures: S. R. Gunn 1998)

# Example: Iris dataset, different kernels


 FIGURE 4.2: Separating Setosa with a linear SVC ( $C = \infty$ )

 FIGURE 4.3: Separating Virginica with a polynomial SVM (degree 2,  $C = \infty$ )

 FIGURE 4.6: Separating Virginica with a polynomial SVM (degree 2,  $C = 10$ )

# Example: Iris dataset, soft-margin: effect of $C$


 (a)  $C = \infty$ 

 (b)  $C = 1000$ 

 (c)  $C = 100$ 

 (d)  $C = 10$ 

 (e)  $C = 1$ 

 (f)  $C = 0.1$



# The Kernel approach

- ▶ rewrite the learning algorithm
- ▶ such that any reference to the input data happens from within inner products
- ▶ replace any such inner product by the kernel function
- ▶ work with the (linear) algorithm as usual
  
- ▶ many well-known algorithms can be rewritten using the kernel approach

## Summary: Kernels

- ▶ non-linearity enters (only) through the kernel
- ▶ but the training algorithm remains linear
  
- ▶ free choice of the kernel (and feature map)
- ▶ based on the application
- ▶ polynomial or Gaussian kernels often work well
  
- ▶ some examples of fancy kernels next week



## Summary: Support Vector Machine

- ▶ based on the linear classifier

Four new main concepts:

- ▶ maximum margin classification
- ▶ soft-margin classification for noisy data
- ▶ introduce non-linearity via feature maps
- ▶ kernel-trick: implicit calculation of feature maps
  
- ▶ use Quadratic Programming for training
- ▶ polynomial or Gaussian kernels often work well