Support Vector Machines

64-360 Algorithmic Learning, part 3a

Norman Hendrich

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

08/06/2011

Planning and overview: AL part 3

- 08/06/2011 support vector machines (1)
  15/06/2011 holidays/KIF
- 22/06/2011 support vector machines (2)
- 29/06/2011 function approximation
- 06/07/2011 reinforcement learning (1)
- 13/07/2011 reinforcement learning (2)

Outline

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 σ

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

- during training, find the *support-vectors* $x_1 \ldots x_n$
- and weights ϑ 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 σ
- get the classification result

Block-diagram

Example: handwritten digit recognition

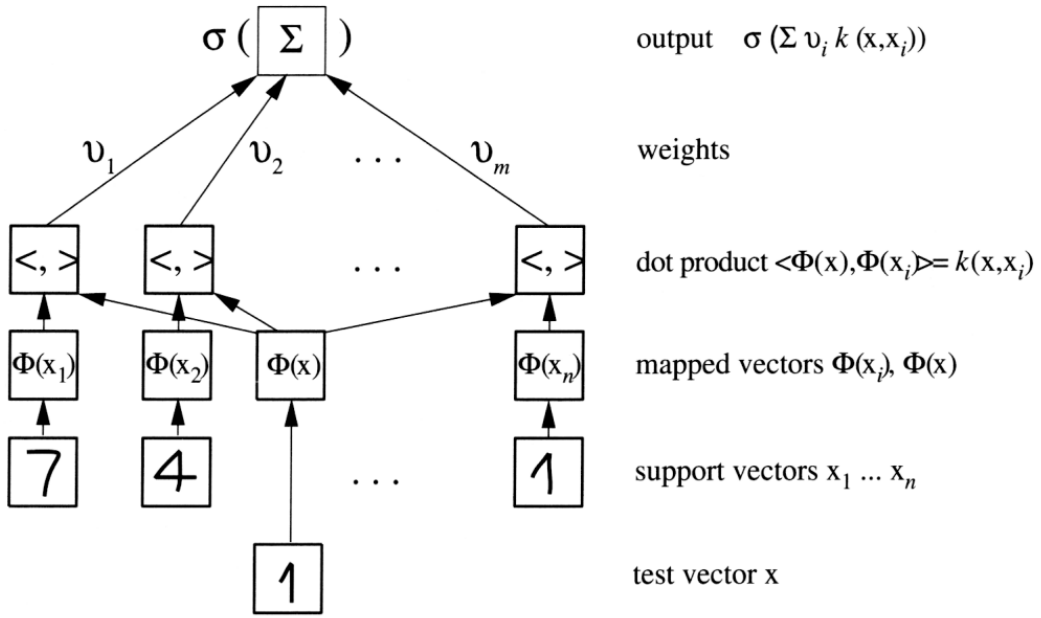
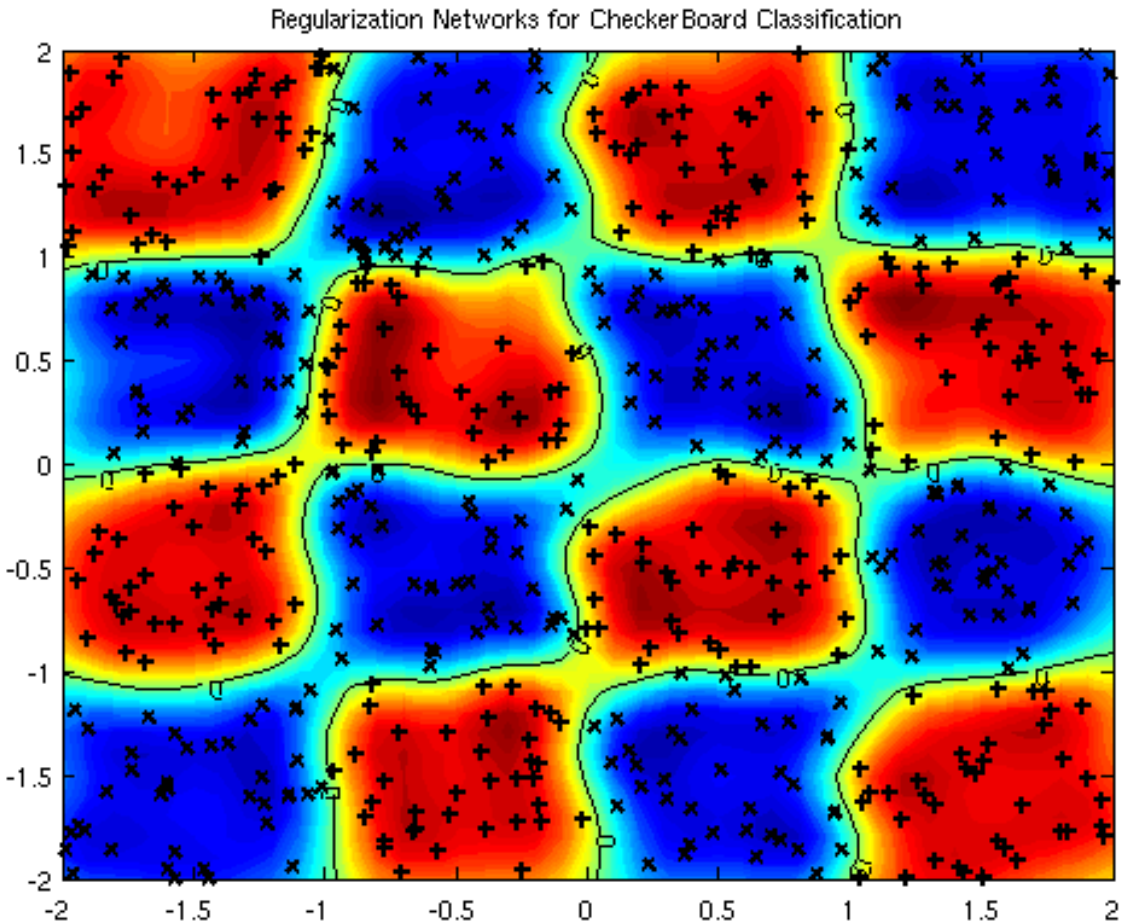


Figure 1.9 Architecture of SVMs and related kernel methods. The input x and the expansion patterns (SVs) x_i (we assume that we are dealing with handwritten digits) are nonlinearly mapped (by Φ) into a feature space $\mathcal{H}$ where dot products are computed. Through the use of the kernel k, these 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 σ (in pattern recognition, $\sigma(x) = \mathrm{sgn}(x + b)$; 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 & Cristianni 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/

- A. W. Moore, *Support Vector Machines*, 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)

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 $y = \mathrm{sgn}(\sum_i w_i \cdot x_i + b)$

Simple classification example

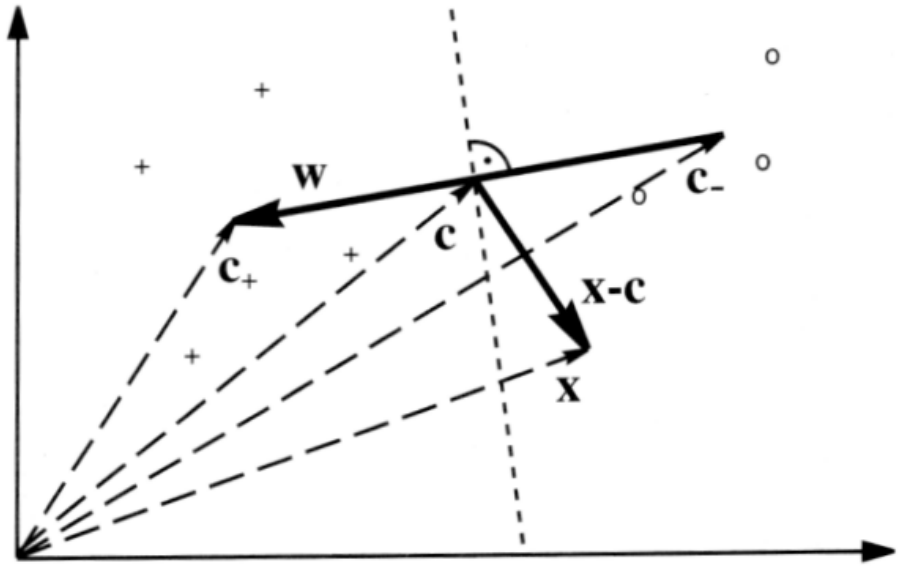
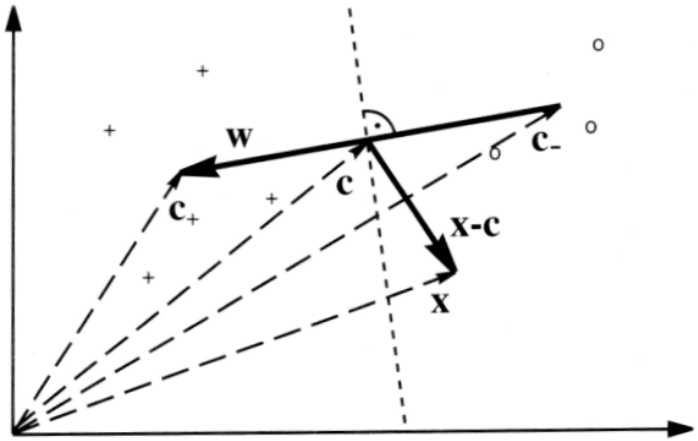


Figure 1.1 A simple geometric classification algorithm: given two classes of points (depicted by 'o' and '+'), compute their means $\mathbf{c}_+, \mathbf{c}_-$ and assign a test pattern $\mathbf{x}$ to the one whose mean is closer. This can be done by looking at the dot product between $\mathbf{x} - \mathbf{c}$ (where $\mathbf{c} = (\mathbf{c}_+ + \mathbf{c}_-)/2$) and $\mathbf{w} := \mathbf{c}_+ - \mathbf{c}_-$, 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 $\mathbf{w}$.

Simple classification example (cont'd)



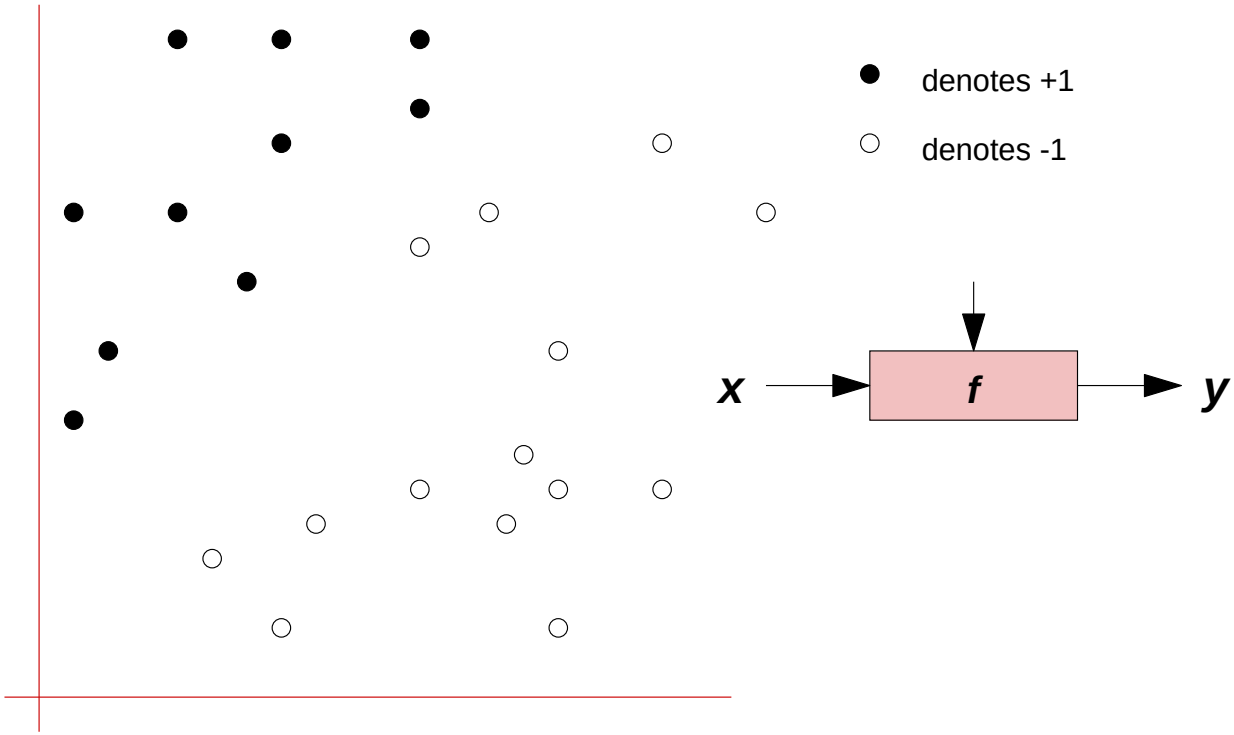
- two classes of data points ('o' and '+')
- calculate the means of each cluster (center of mass)
- assign test pattern x to the nearest cluster
- can be written as $y = \mathrm{sgn}(\sum_{i=1}^{m} \alpha_i \langle x, x_i \rangle + b)$
- with constant weights $\alpha_i = \{\frac{1}{m_+}, \frac{1}{m_-}\}$

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: $y = \mathrm{sgn}\langle (x - c), w \rangle$
- norm: $||x|| := \sqrt{\langle x, x \rangle}$
- rewrite: $y = \mathrm{sgn}(\langle (x, c_+) \rangle - \langle (x, c_-) \rangle + b)$
  with $b = (||c_-||^2 - ||c_+||^2)/2$
- all together:

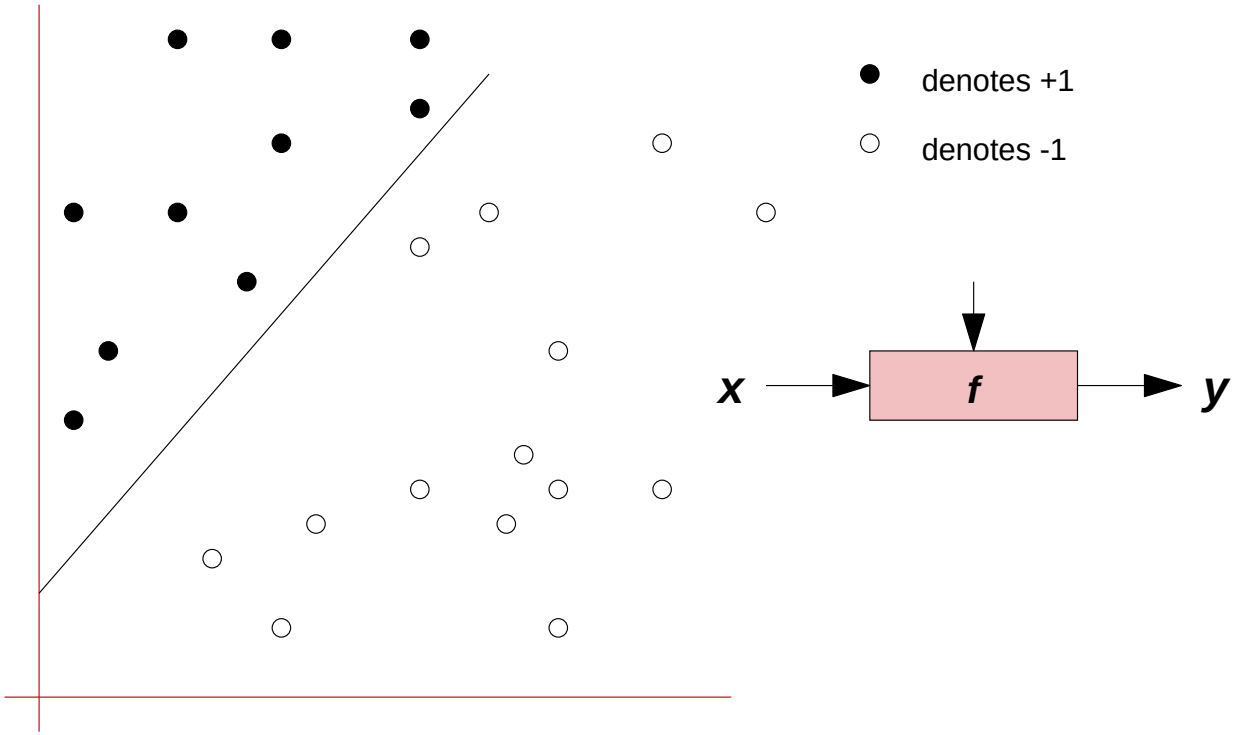
$$y = \mathrm{sgn}\Big(\frac{1}{m_+} \sum_{\{i|y_i=+1\}} x_i \langle x, x_i \rangle - \frac{1}{m_-} \sum_{\{i|y_i=-1\}} x_i \langle x, x_i \rangle + b\Big)$$

Linear classification



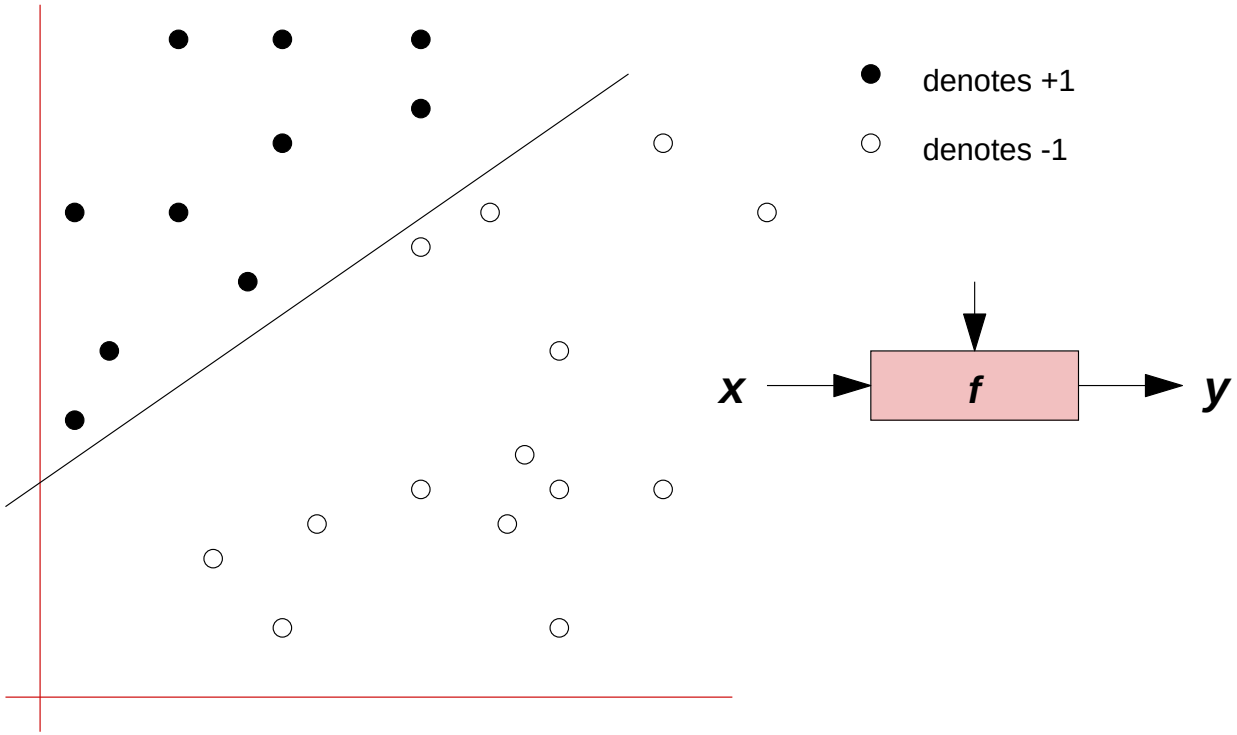
- find w and b, so that $y(x, w, b) = \mathrm{sgn}(w \cdot x - b)$

Linear classification



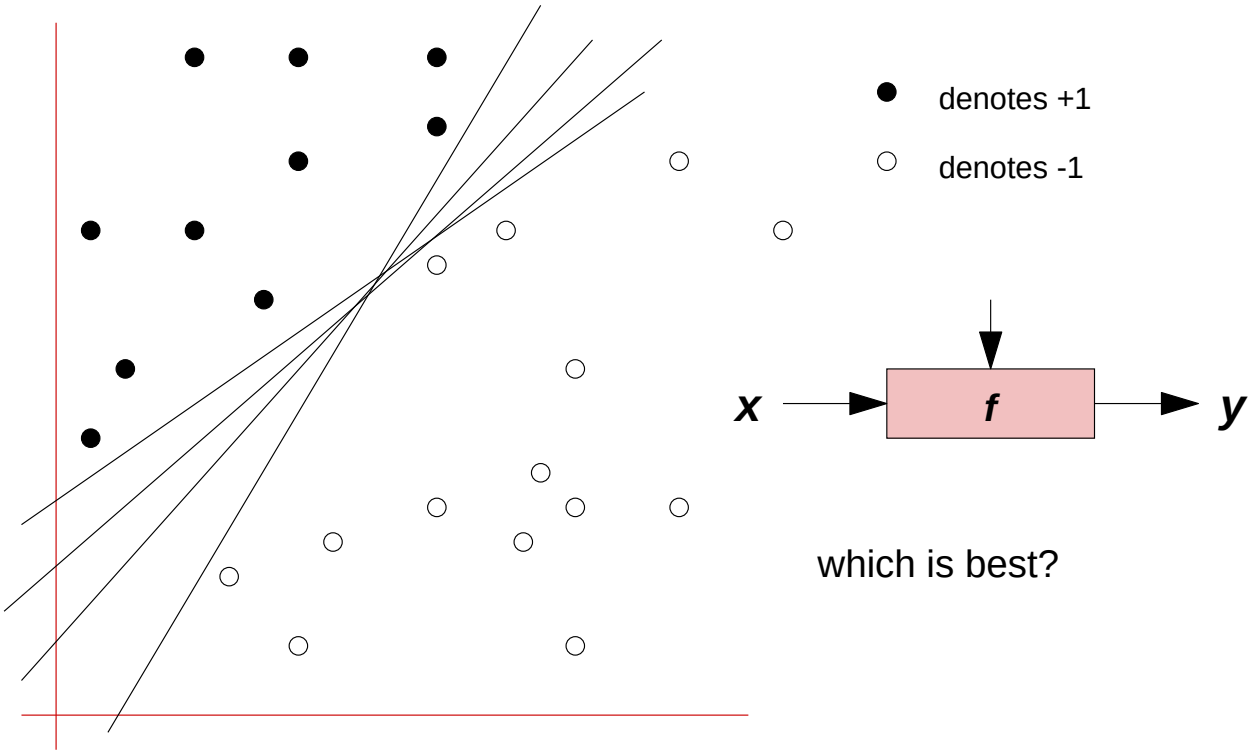
- one possible decision boundary

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

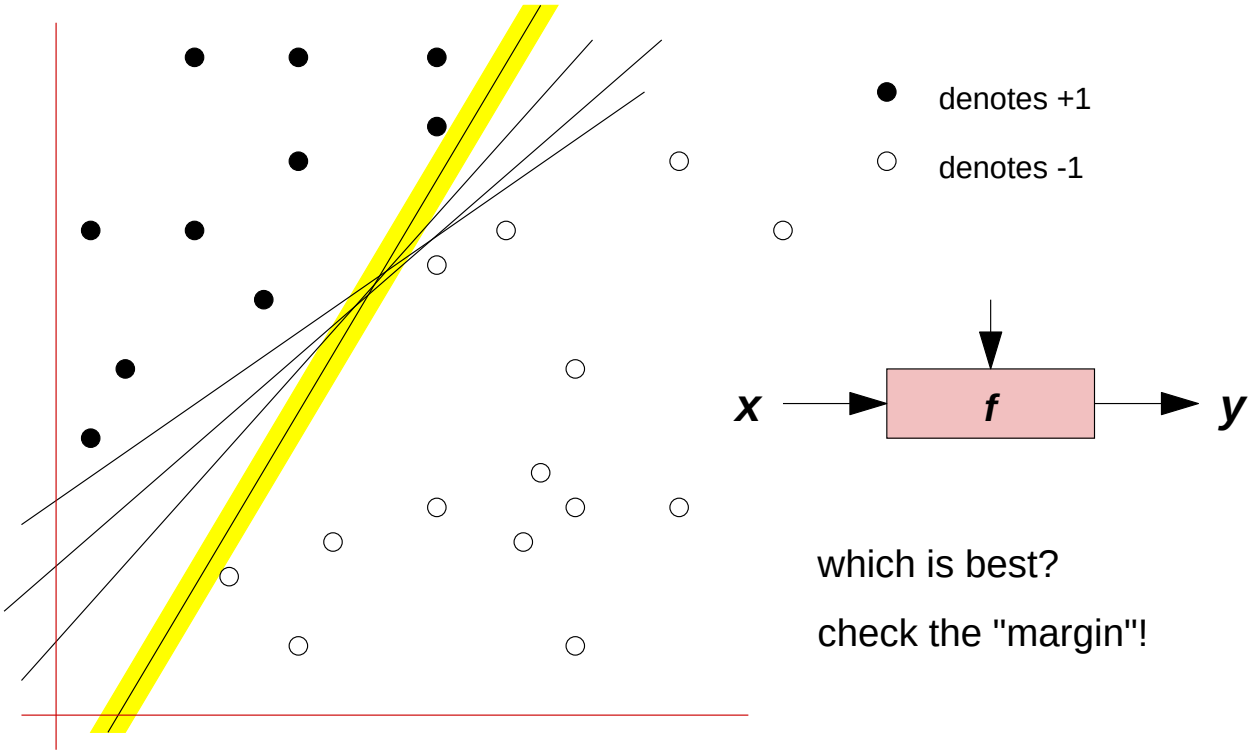
Perceptron training algorithm

Input: data $\mathcal{S}$

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

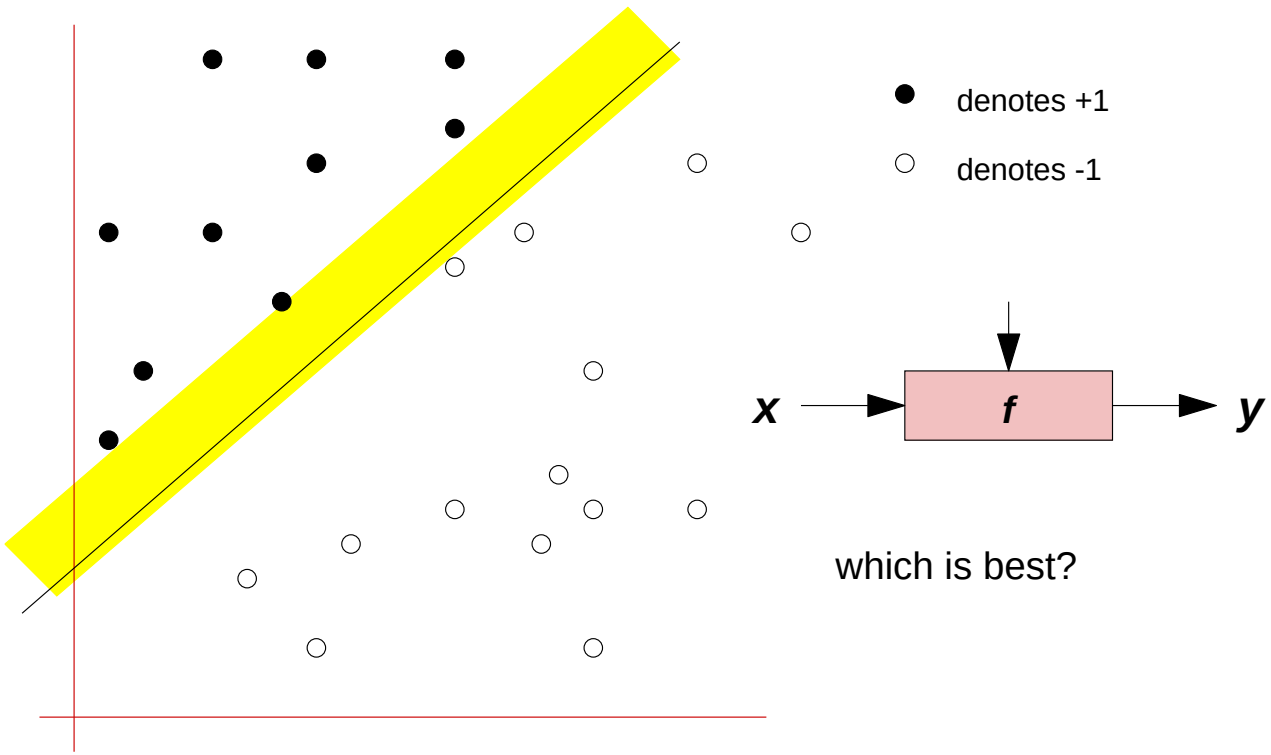
1: **repeat**
2: $err \leftarrow 0$
3: **for** $i = 1, \ldots, \ell$ **do**
4: compute $f(x_i) = \text{sign}(\langle w, \phi(x_i) \rangle + b)$
5: **if** $f(x_i) \neq y_i$ **then**
6: $w \leftarrow w + y_i \phi(x_i)$
7: $b \leftarrow b + y_i$
8: $err = err + 1$
9: **end if**
10: **end for**
11: **until** $err = 0$
12: **return** w, b

The classifier margin



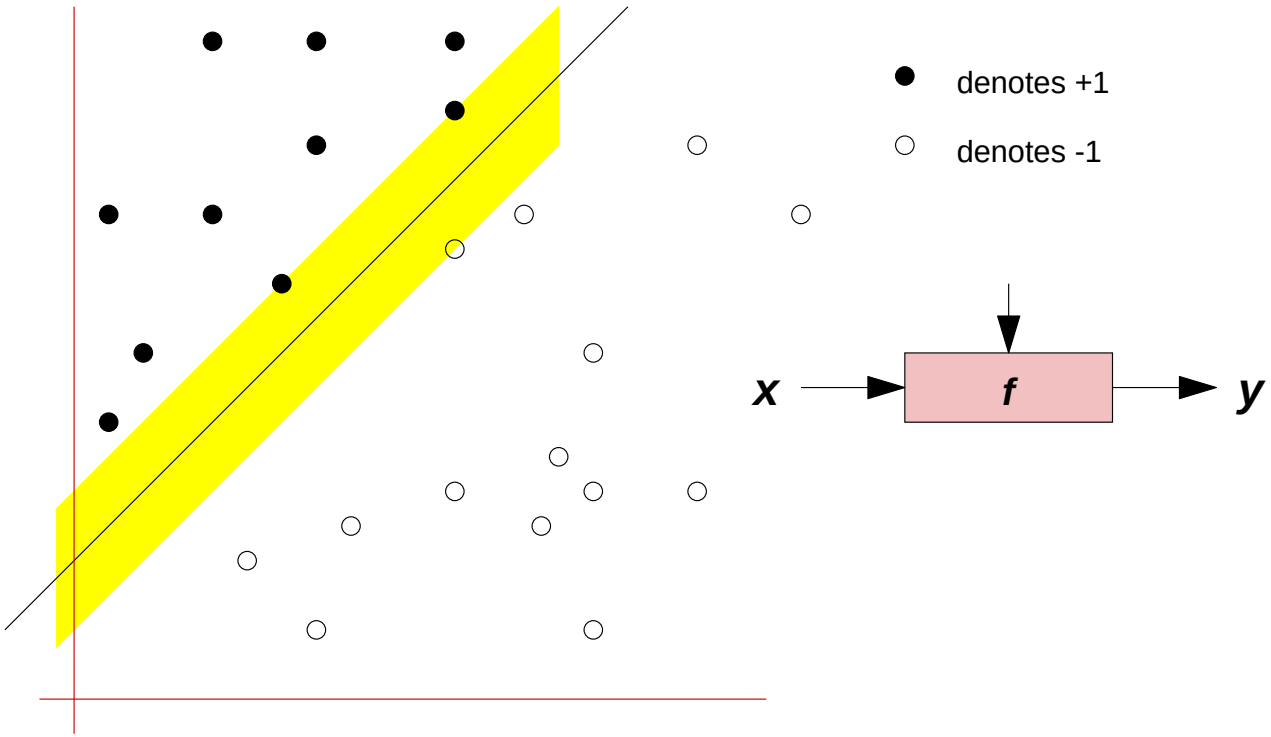
- define the *margin* as the width that the boundary could be increased before hitting a data point.

The classifier margin



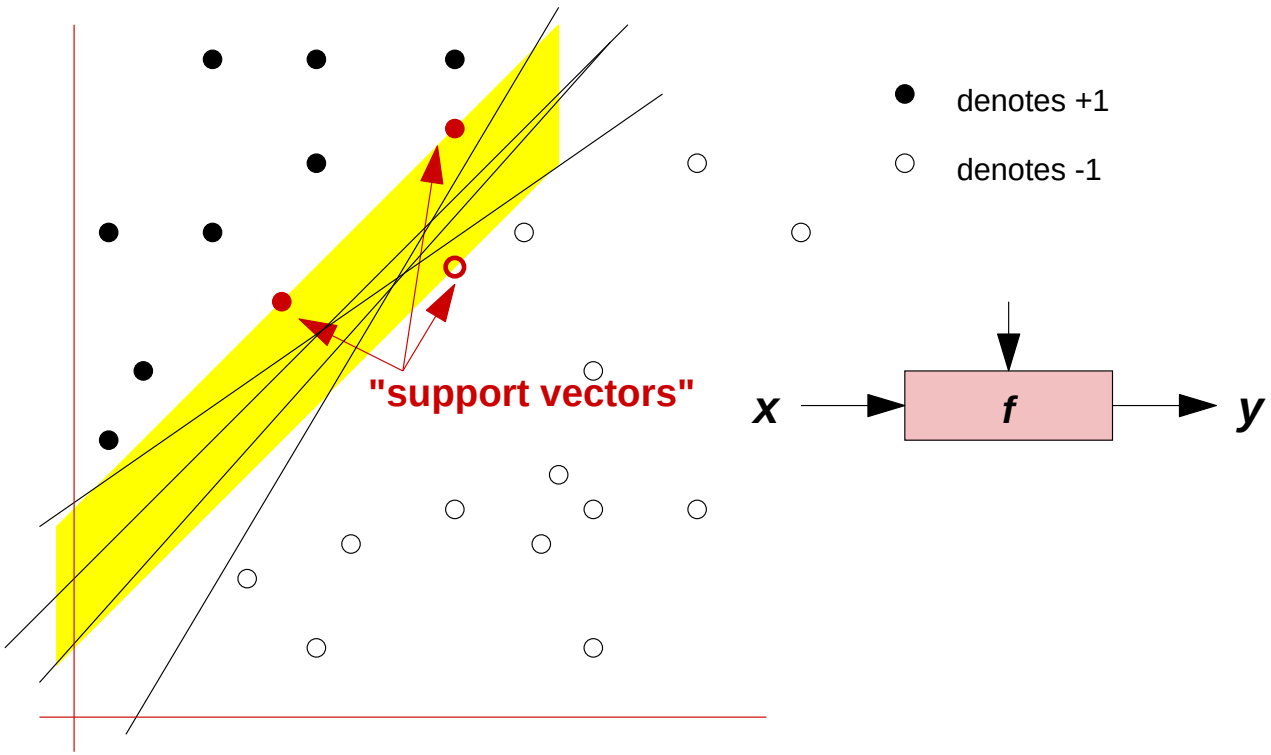
- a second example: margin not symmetrical

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

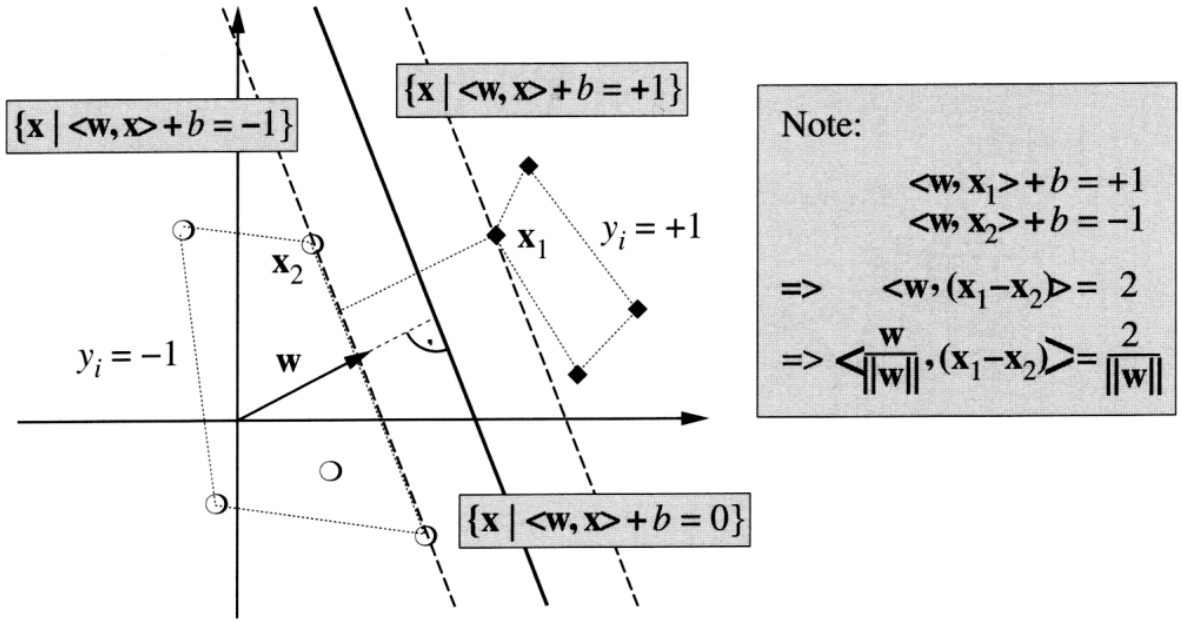
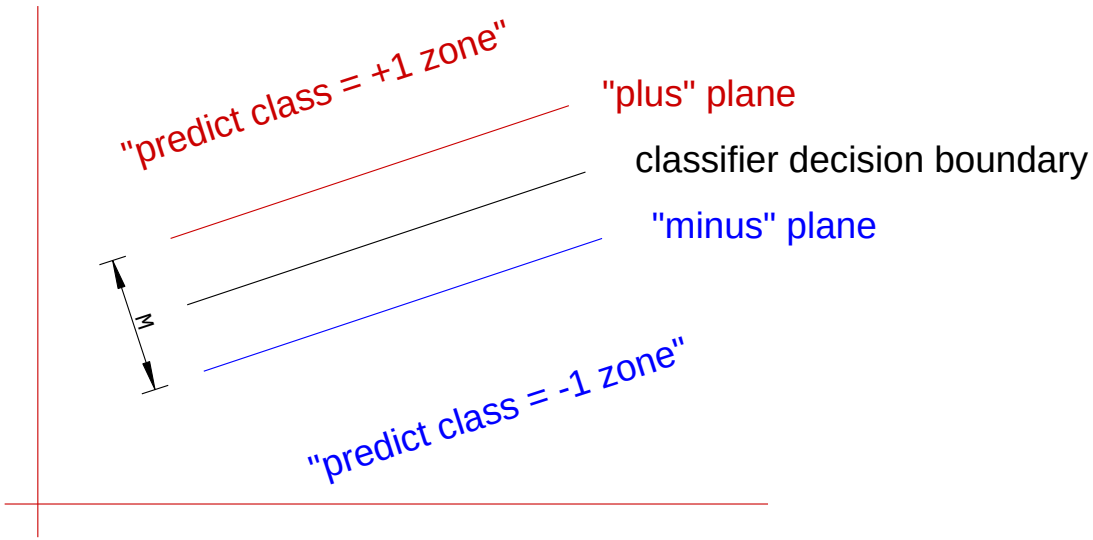


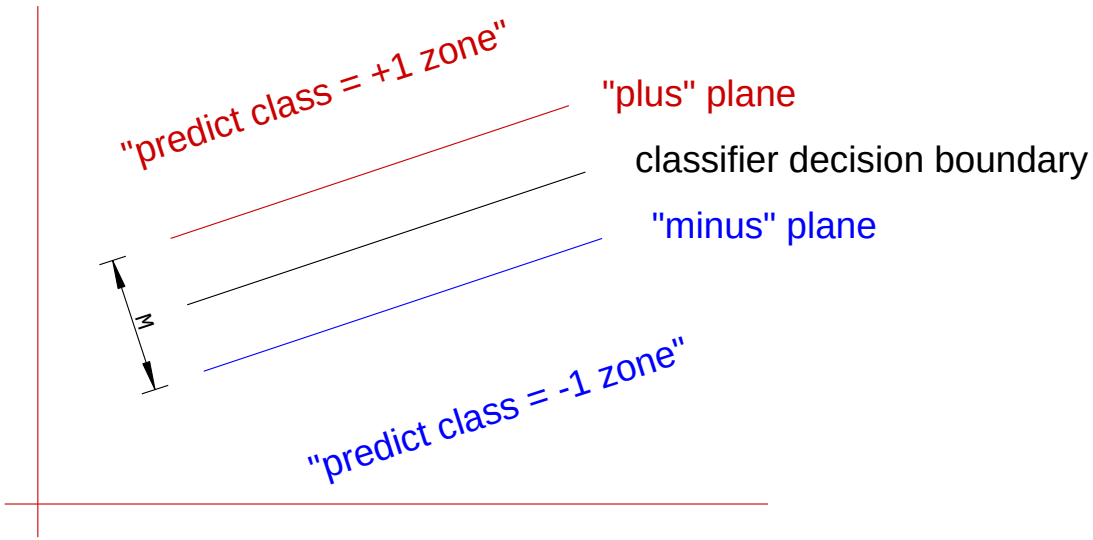
Figure 1.5 A binary classification toy problem: separate balls from diamonds. The *optimal hyperplane* (1.23) is shown as a solid line. The problem being separable, there exists a weight vector $\mathbf{w}$ and a threshold b such that $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) > 0$ $(i = 1, \ldots, m)$. 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



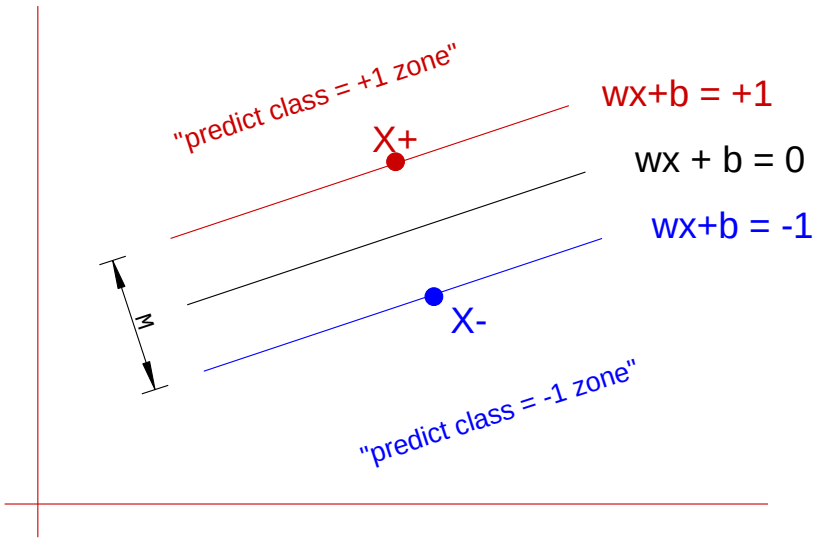
- how to represent the boundary (hyperplane)
- and the margin width M
- in m input dimensions?

Calculating the margin width



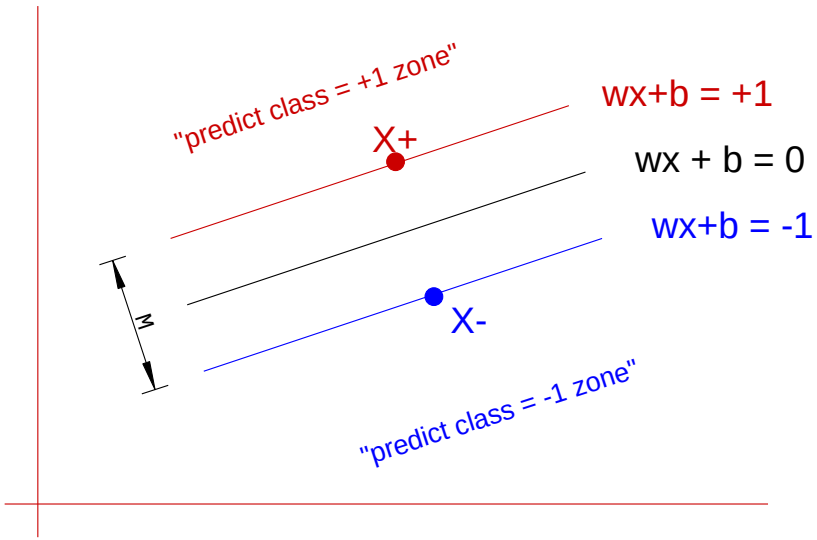
- plus-plane: $\{x : w \cdot x + b = +1\}$
- minus-plane: $\{x : w \cdot x + b = -1\}$
- classify pattern as $+1$ if $w \cdot x + b \geq +1$
  and -1 if $w \cdot x + b \leq -1$

Calculating the margin width



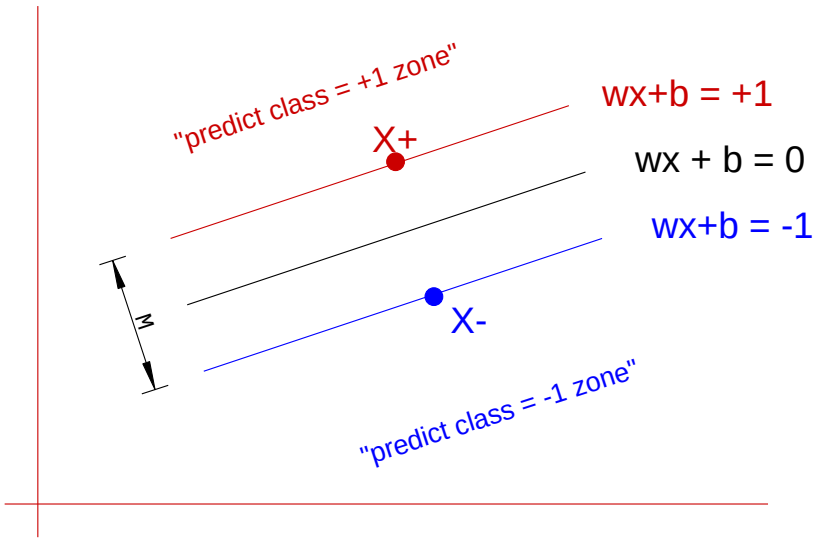
- w is perpendicular to the decision boundary
- and the plus-plane and minus-plane
- proof: consider two points u and v on the plus-plane and calculate $w \cdot (u - v)$

Calculating the margin width



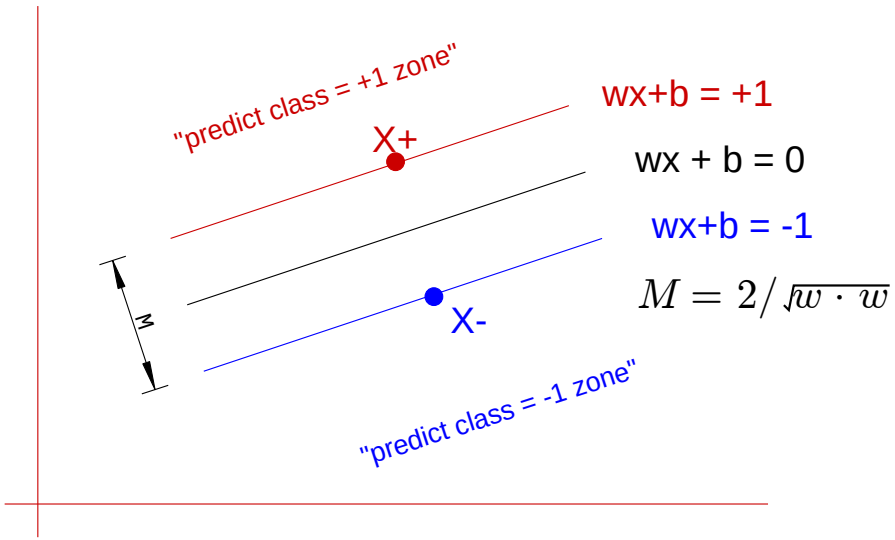
- select point X^+ on the plus plane
- and nearest point X^- on the minus plane
- of course, margin width $M = |X^+ - X^-|$
- and $X^+ = X^- + \lambda w$ for some λ

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, Quadrating 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
- e.g. LOQO http://www.princeton.edu/~rvdb/loqo

- or algorithms streamlined for SVM (e.g. large data sets)

Quadratic Programming

General problem:

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

- subject to n linear inequality constraints

$$a_{11} u_1 + a_{12} u_2 + \cdots + a_{1m} u_m \leq b_1$$
$$a_{21} u_1 + a_{22} u_2 + \cdots + a_{2m} u_m \leq b_2$$
$$\ldots$$
$$a_{n1} u_1 + a_{n2} u_2 + \cdots + a_{nm} u_m \leq b_n$$

- subject to e additional linear equality constraints

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

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:

  $w \cdot x_k + b \geq +1 \quad \text{if } y_k = +1$

  $w \cdot x_k + b \leq -1 \quad \text{if } y_k = -1$

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 α_i
- this allows to rewrite with simple inequalities $\alpha_i \geq 0$
- solve the optimization problem, find α_i
- from the α_i, find the separating hyperplane (w)
- from the hyperplane, find b

The dual optimization problem

Directly solving this problem is difficult because the constraints are quite complex. The mathematical tool of choice for simplifying this problem is the Lagrangian duality theory (e.g., Bertsekas, 1995). This approach leads to solving the following dual problem:

$$\begin{aligned} &\max \ \mathcal{D}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i \alpha_i \, y_j \alpha_j \, \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}_j) \\ &\text{subject to} \quad \begin{cases} \forall i \quad \alpha_i \geq 0, \\ \sum_i y_i \alpha_i = 0. \end{cases} \end{aligned} \tag{1.2}$$

Problem (1.2) is computationally easier because its constraints are much simpler. The direction $\mathbf{w}^*$ of the optimal hyperplane is then recovered from a solution $\boldsymbol{\alpha}^*$ of the dual optimization problem (1.2).

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

Determining the bias b^* becomes a simple one-dimensional problem. The linear discriminant function can then be written as

$$\hat{y}(\mathbf{x}) = \mathbf{w}^{*\top} \mathbf{x} + b^* = \sum_{i=1}^{n} y_i \alpha_i \, \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}) + b^*. \tag{1.3}$$

Dual representation

- In simple Perceptron training, the weight vector w can always be rewritten as a linear combination of training data points:

$$w = \sum_{i=1}^{\ell} \alpha_i y_i \phi(x_i).$$

6: $w \leftarrow w + y_i x_i$

- Coefficient α_i says how often x_i was misclassified.
- Indirect evaluation of dot product with weight vector becomes possible without explicitly representing it.

$$\begin{aligned} \langle w, \phi(x) \rangle &= \left\langle \sum_{i=1}^{\ell} \alpha_i y_i \phi(x_i), \phi(x) \right\rangle \\ &= \sum_{i=1}^{\ell} \alpha_i y_i \langle \phi(x_i), \phi(x) \rangle . \end{aligned}$$

Dual representation of Perceptron learning

Algorithm 1 Perceptron Training (dual form)

Initialize: $\alpha_1, \ldots, \alpha_\ell \leftarrow 0$, $b \leftarrow 0$

1: **repeat**
2: $err \leftarrow 0$
3: **for** $i = 1, \ldots, \ell$ **do**
4: $\gamma_i = y_i(\sum_{j=1}^{\ell} \alpha_j y_j \langle \phi(x_j), \phi(x_i) \rangle + b)$
5: **if** $\gamma_i \leq 0$ **then**
6: $\alpha_i \leftarrow \alpha_i + 1$
7: $b \leftarrow b + y_i$
8: $err = err + 1$
9: **end if**
10: **end for**
11: **until** $err = 0$
12: **return** $\alpha_1, \ldots, \alpha_\ell, b$

the only reference to $\phi(x)$ and only for the known data items.

rewritten

rewritten

rewritten

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?

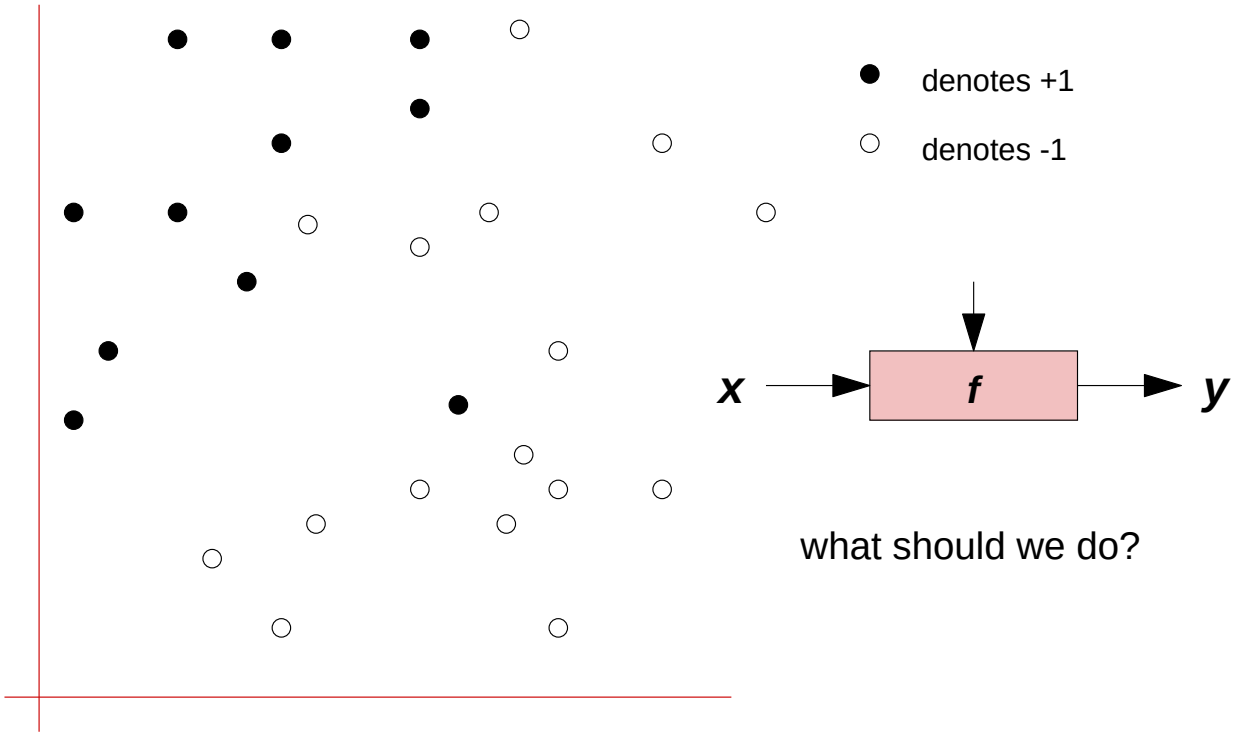
- actual "real world" training data contains noise
- usually, several "outlier" patterns
- for example, mis-classified training data

- 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

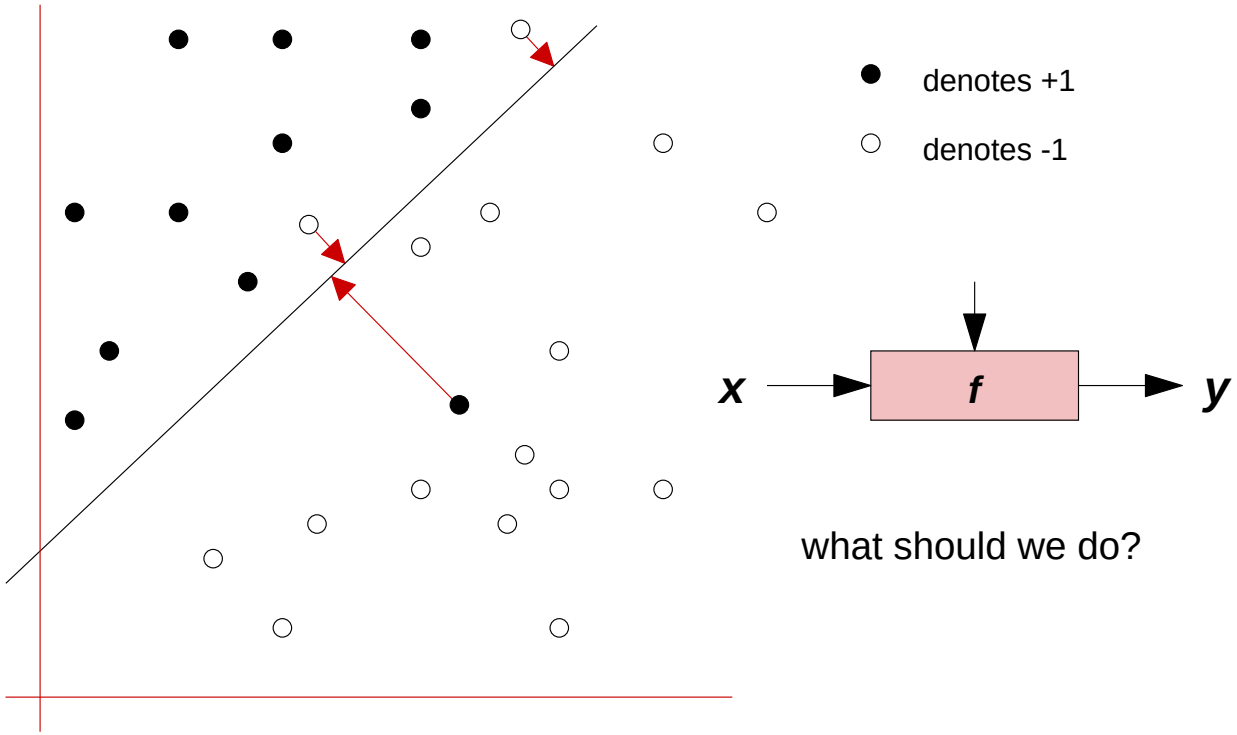
- instead, allow some patterns to violate the margin constraints

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

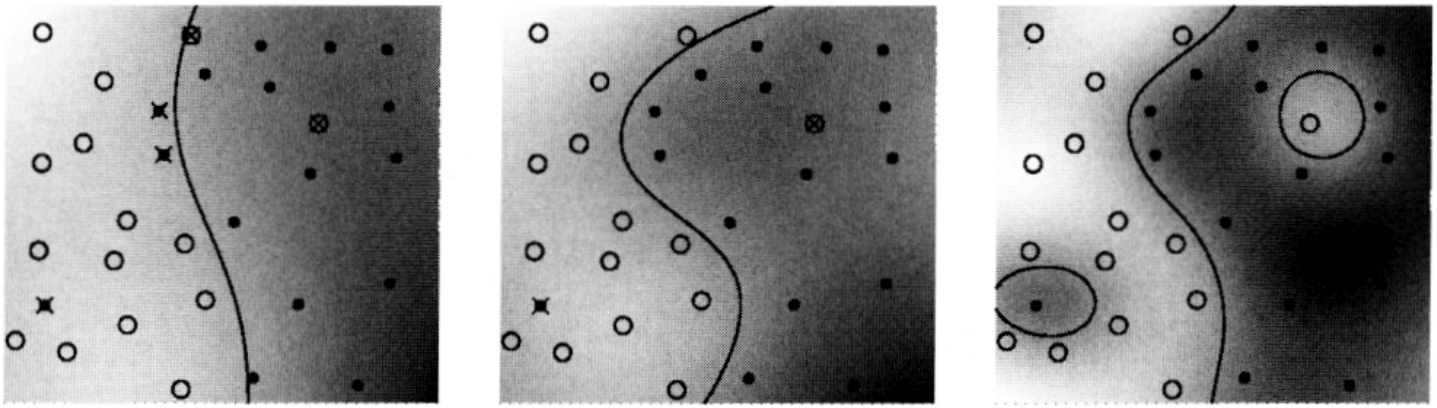


Figure 1.2 2D toy example of binary classification, solved using three models (the decision boundaries are shown). The models vary in complexity, ranging from a simple one *(left)*, which misclassifies a large number of points, to a complex one *(right)*, which "trusts" each point and comes up with solution 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 $\xi = (\xi_i \dots \xi_n)$, $\xi_i \geq 0$
- 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 ξ_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:

$$\forall i : \quad y(x_i)(w \cdot \Phi(x_i) + b) \geq 1 - \xi_i$$
$$\forall i : \quad \xi_i \geq 0$$

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

Slack parameters ξ_i, control parameter C

(LSKM chapter 1)

Optimal hyperplanes (section 1.2.1) are useless when the training set is not linearly separable. Kernel machines (section 1.2.2) can represent complicated decision boundaries that accomodate any training set. But this is not very wise when the problem is very noisy.

Cortes and Vapnik (1995) show that noisy problems are best addressed by allowing some examples to violate the margin constraints in the primal problem (1.1). These potential violations are represented using positive slack variables $\boldsymbol{\xi} = (\xi_i \dots \xi_n)$. An additional parameter C controls the compromise between large margins and small margin violations.

$$\begin{aligned} \max_{\mathbf{w},b,\boldsymbol{\xi}} \ & \mathcal{P}(\mathbf{w}, b, \boldsymbol{\xi}) = \frac{1}{2}\mathbf{w}^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \ & \begin{cases} \forall i & y_i(\mathbf{w}^\top \Phi(\mathbf{x}_i) + b) \geq 1 - \xi_i \\ \forall i & \xi_i \geq 0 \end{cases} \end{aligned} \tag{1.4}$$

The dual formulation of this soft-margin problem is strikingly similar to the dual formulation (1.2) of the optimal hyperplane algorithm. The only change is the appearance of the upper bound C for the coefficients $\boldsymbol{\alpha}$.

Lagrange formulation of the soft-margin SVM

The difficulty of the primal problem (1.4) lies with the complicated inequality constraints that represent the margin condition. We can represent these constraints using positive Lagrange coefficients $\alpha_i \geq 0$.

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i\left(y_i(\mathbf{w}^\top\Phi(\mathbf{x}_i) + b) - 1 + \xi_i\right).$$

The formal dual objective function $\underline{\mathcal{D}}(\boldsymbol{\alpha})$ is defined as

$$\underline{\mathcal{D}}(\boldsymbol{\alpha}) \quad = \quad \min_{\mathbf{w},b,\boldsymbol{\xi}} L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}) \quad \text{subject to} \quad \forall i \;\; \xi_i \geq 0. \tag{1.8}$$

This minimization no longer features the complicated constraints expressed by the Lagrange coefficients. The $\xi_i \geq 0$ constraints have been kept because they are easy enough to handle directly. Standard differential arguments[1] yield the analytical expression of the dual objective function.

$$\underline{\mathcal{D}}(\boldsymbol{\alpha}) = \begin{cases} \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} y_i\alpha_i\, y_j\alpha_j\, K_{ij} & \text{if } \sum_i y_i\alpha_i = 0 \text{ and } \forall i\ \alpha_i \leq C, \\ -\infty & \text{otherwise.} \end{cases}$$

Dual formulation of soft-margin SVM

The rest of this chapter focuses on solving the soft-margin SVM problem (1.4) using the standard dual formulation (1.5),

$$\max \ \mathcal{D}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i \alpha_i \, y_j \alpha_j \, K_{ij}$$
$$\text{subject to} \begin{cases} \forall i \quad 0 \le \alpha_i \le C, \\ \sum_i y_i \alpha_i = 0, \end{cases}$$

where $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ is the matrix of kernel values.

After computing the solution $\boldsymbol{\alpha}^*$, the SVM discriminant function is

$$\hat{y}(\mathbf{x}) = \mathbf{w}^{*\top} \mathbf{x} + b^* = \sum_{i=1}^{n} \alpha_i^* K(\mathbf{x}_i, \mathbf{x}) + b^*. \tag{1.6}$$

The optimal bias b^* can be determined by returning to the primal problem, or, more efficiently, using the optimality criterion (1.11) discussed below.

It is sometimes convenient to rewrite the box constraint $0 \le \alpha_i \le C$ as a box constraint on the quantity $y_i \alpha_i$:

$$y_i \alpha_i \in [A_i, B_i] = \begin{cases} [\,0, C\,] & \text{if } y_i = +1, \\ [\,-C, 0\,] & \text{if } y_i = -1. \end{cases} \tag{1.7}$$

The optimization problem

$$\max_{\boldsymbol{\alpha}'} \mathcal{D}(\boldsymbol{\alpha}') = \sum_{i=1}^{n} \alpha_i' - \frac{1}{2} \sum_{i,j=1}^{n} y_i \alpha_i' \, y_j \alpha_j' \, K(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{subject to} \begin{cases} \forall i \notin \mathcal{B} & \alpha_i' = \alpha_i, \\ \forall i \in \mathcal{B} & 0 \leq \alpha_i' \leq C, \\ & \sum_i y_i \alpha_i' = 0. \end{cases} \tag{1.19}$$

We can rewrite (1.19) as a quadratic programming problem in variables α_i, $i \in \mathcal{B}$ and remove the additive terms that do not involve the optimization variables $\boldsymbol{\alpha}'$:

$$\max_{\boldsymbol{\alpha}'} \sum_{i \in \mathcal{B}} \alpha_i' \left(1 - y_i \sum_{j \notin \mathcal{B}} y_j \alpha_j \, K_{ij} \right) - \frac{1}{2} \sum_{i \in \mathcal{B}} \sum_{j \in \mathcal{B}} y_i \alpha_i' \, y_j \alpha_j' \, K_{ij} \tag{1.20}$$

$$\text{subject to } \forall i \in \mathcal{B} \;\; 0 \leq \alpha_i' \leq C \;\text{ and }\; \sum_{i \in \mathcal{B}} y_i \alpha_i' = - \sum_{j \notin \mathcal{B}} y_j \alpha_j.$$

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 ξ_i (slack)
- again, can be written as a QP problem
- again, dual formulation easier to solve

Nonlinearity through feature maps

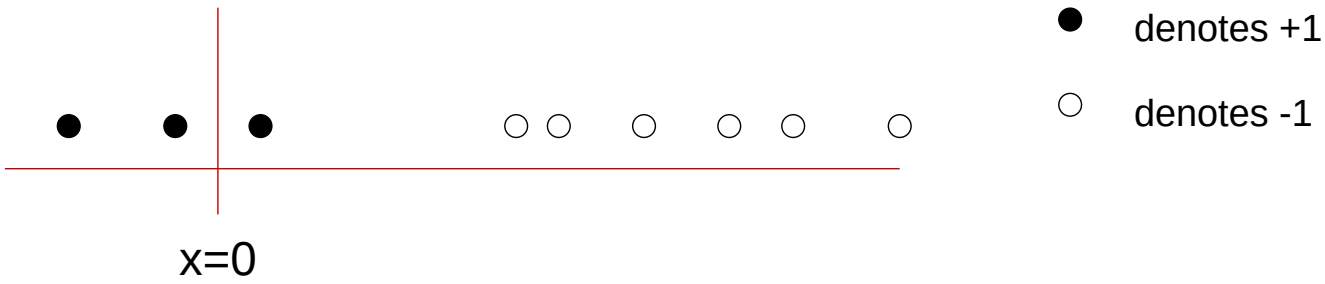
General idea:

- introduce a function Φ which maps the input data into a higher dimensional *feature space*

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

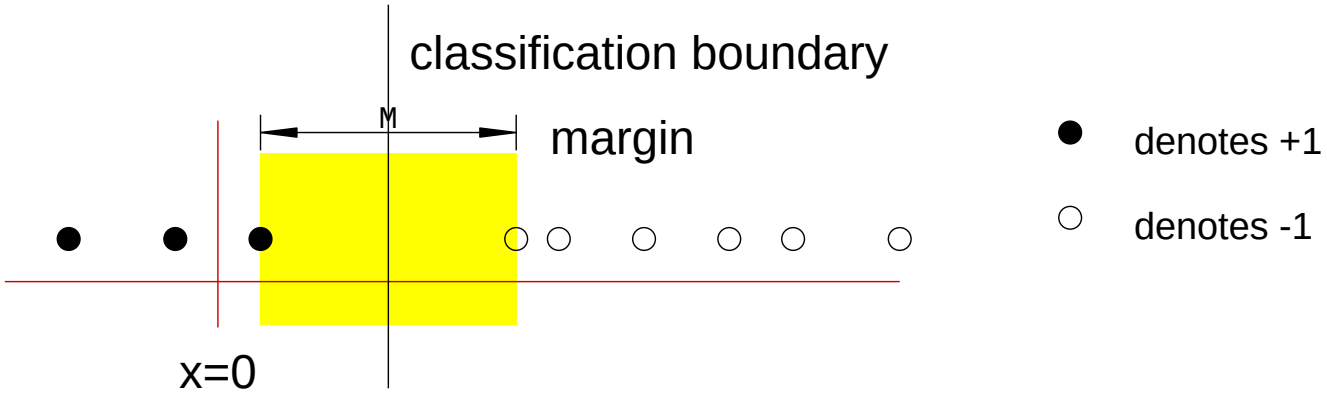
- similar to hidden layers of multi-layer ANNs
- explicit mappings can be expensive in terms of CPU and/or memory (especially in high dimensions)
- "*Kernel functions*" achieve this mapping *implicitly*
- often, very good performance

Example 1-dimensional data set



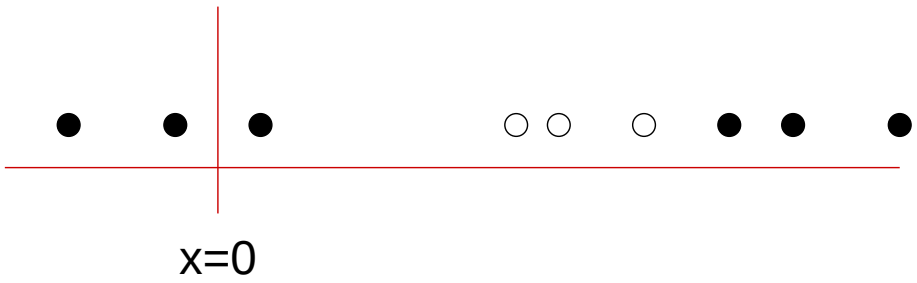
- what would the linear SVM do with these patterns?

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



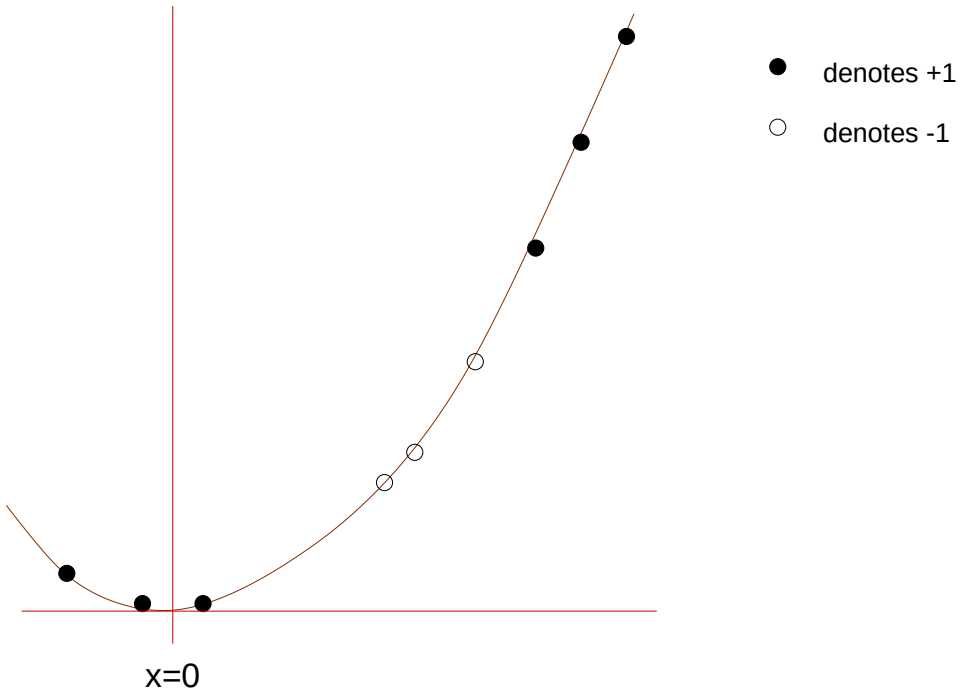
● denotes +1

○ denotes -1

- and now?

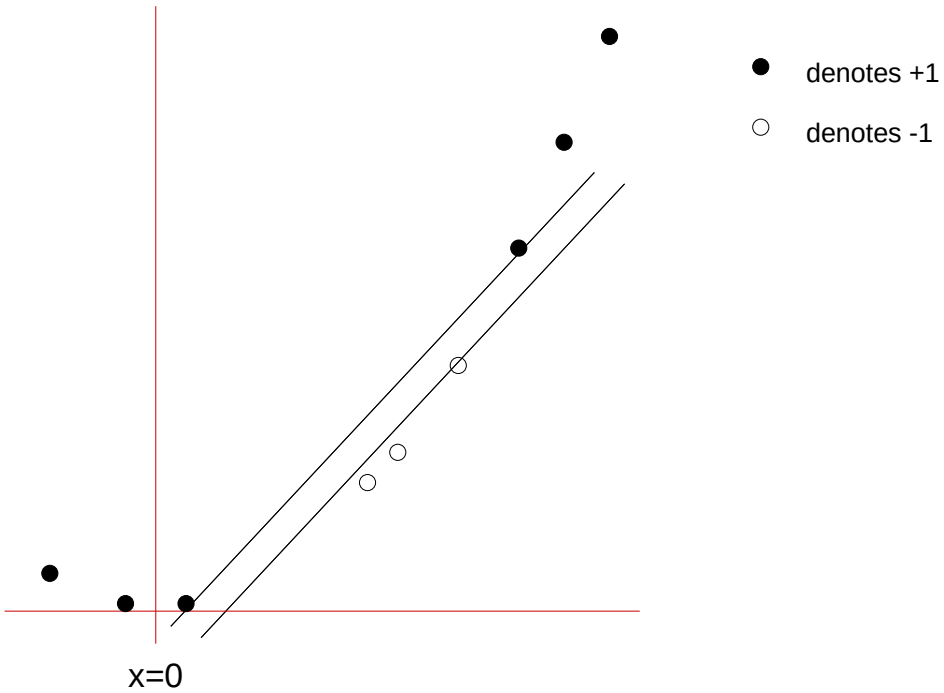
- doesn't look like "outliers"
- so, soft-margin won't help a lot

Harder 1-dimensional data set



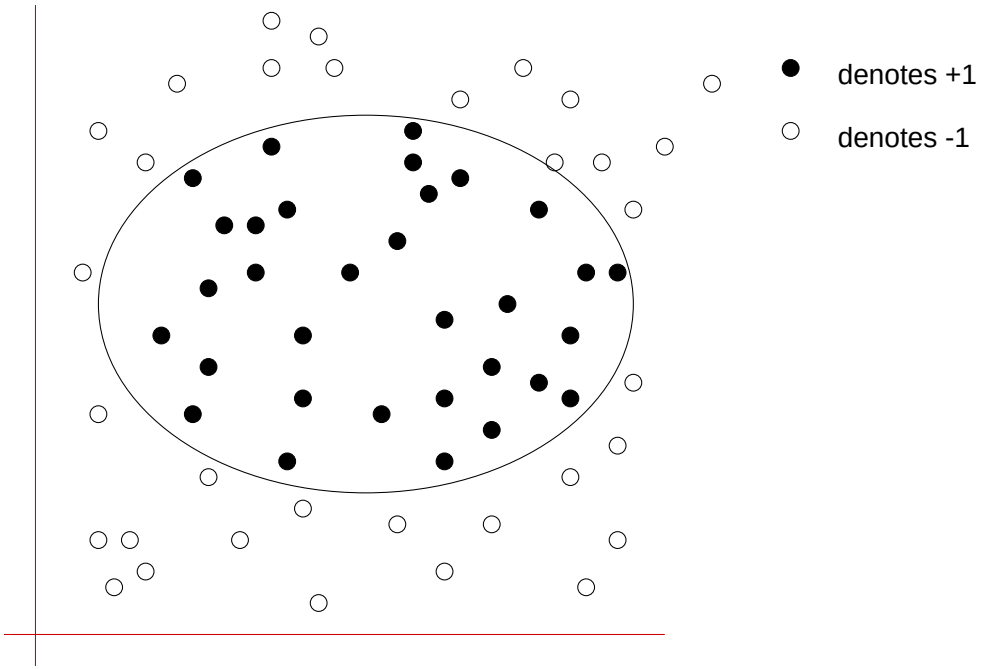
- permit non-linear basis functions
- $z_k = (x_k, x_k^2)$

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

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

Note:

- feature map Φ 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 Φ 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:
  $\Phi'(x) = (x_1^2, x_2^2, x_1x_2, x_2x_1) \in \mathbb{R}^4$
  also fits to $k(x, z) = \langle x, z \rangle^2$

Example: quadratic kernel, m dimensions

- $x = (x_1, \ldots, x_m)$
- $\Phi(x) = ($
  $1,$
  $\sqrt{2}x_1, \sqrt{2}x_2, \ldots \sqrt{2}x_m,$
  $x_1^2, x_2^2, \ldots x_m^2,$
  $\sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \ldots, \sqrt{2}x_{m-1}x_m\ )$

- 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 $(x \cdot y + 1)^2$:

- $$\begin{aligned}(x \cdot y + 1)^2 &= ((\textstyle\sum_{i=1}^{m} x_i \cdot y_i) + 1)^2 \\ &= \left(\textstyle\sum_{i=1}^{m} x_i y_i\right)^2 + 2\left(\textstyle\sum_{i=1}^{m} x_i y_i\right) + 1 \\ &= \textstyle\sum_{i=1}^{m} \sum_{j=1}^{m} x_i y_i x_j y_j + 2 \sum_{i=1}^{m} x_i y_i + 1 \\ &= \textstyle\sum_{i=1}^{m} (x_i y_i)^2 + 2 \sum_{i=1}^{m} \sum_{j=1}^{m} x_i y_i x_j y_j + 2 \sum_{i=1}^{m} x_i y_i + 1 \\ &= \Phi(x) \cdot \Phi(y)\end{aligned}$$

- 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 enourmous 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 make $(a\langle x, y\rangle + b)$ lie between ± 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

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

- with "bandwidth" parameter σ
- kernel evaluation depends on distance of x and z

- *local neighborhood* classification
- initialize σ to a characteristic distance between nearby patterns in feature space
- large distance implies orthogonal patterns

The Kernel "Trick"

- 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