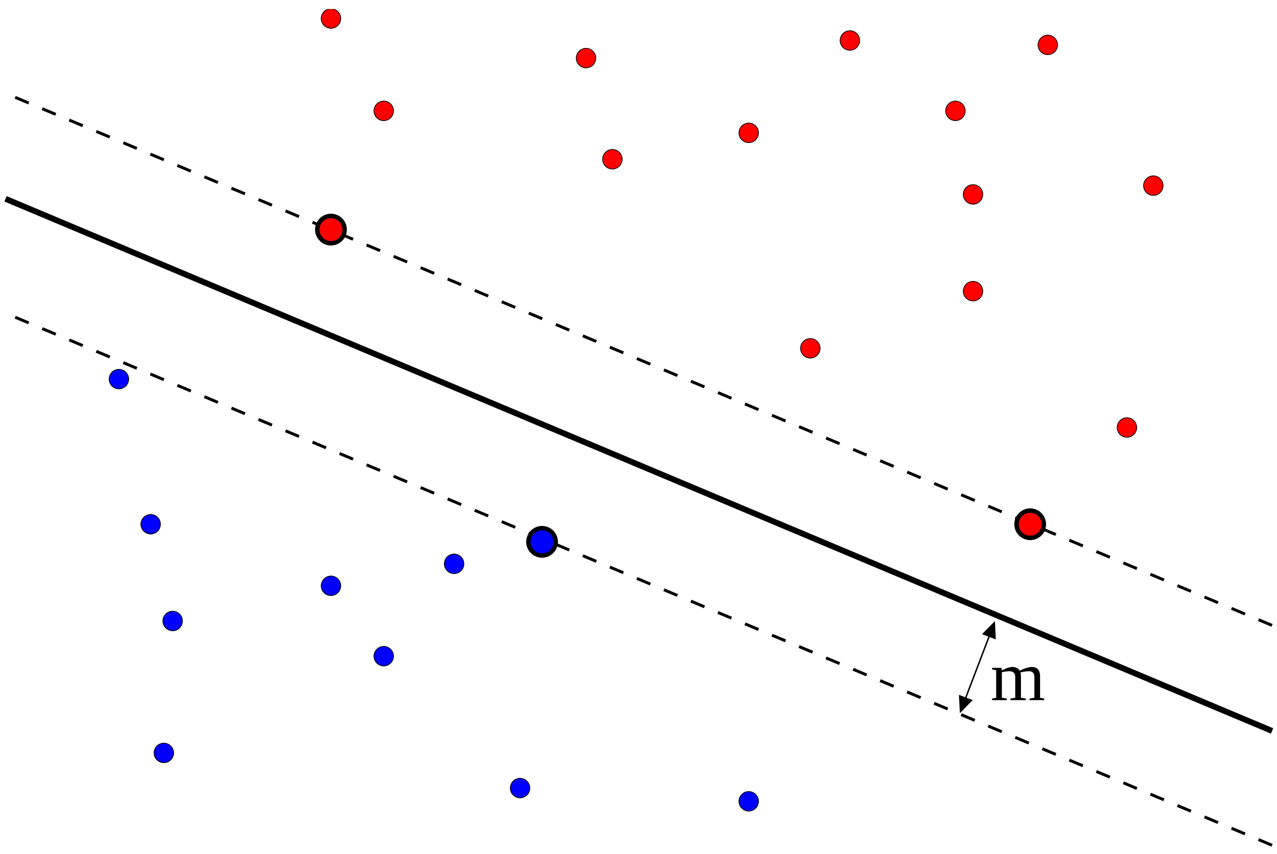
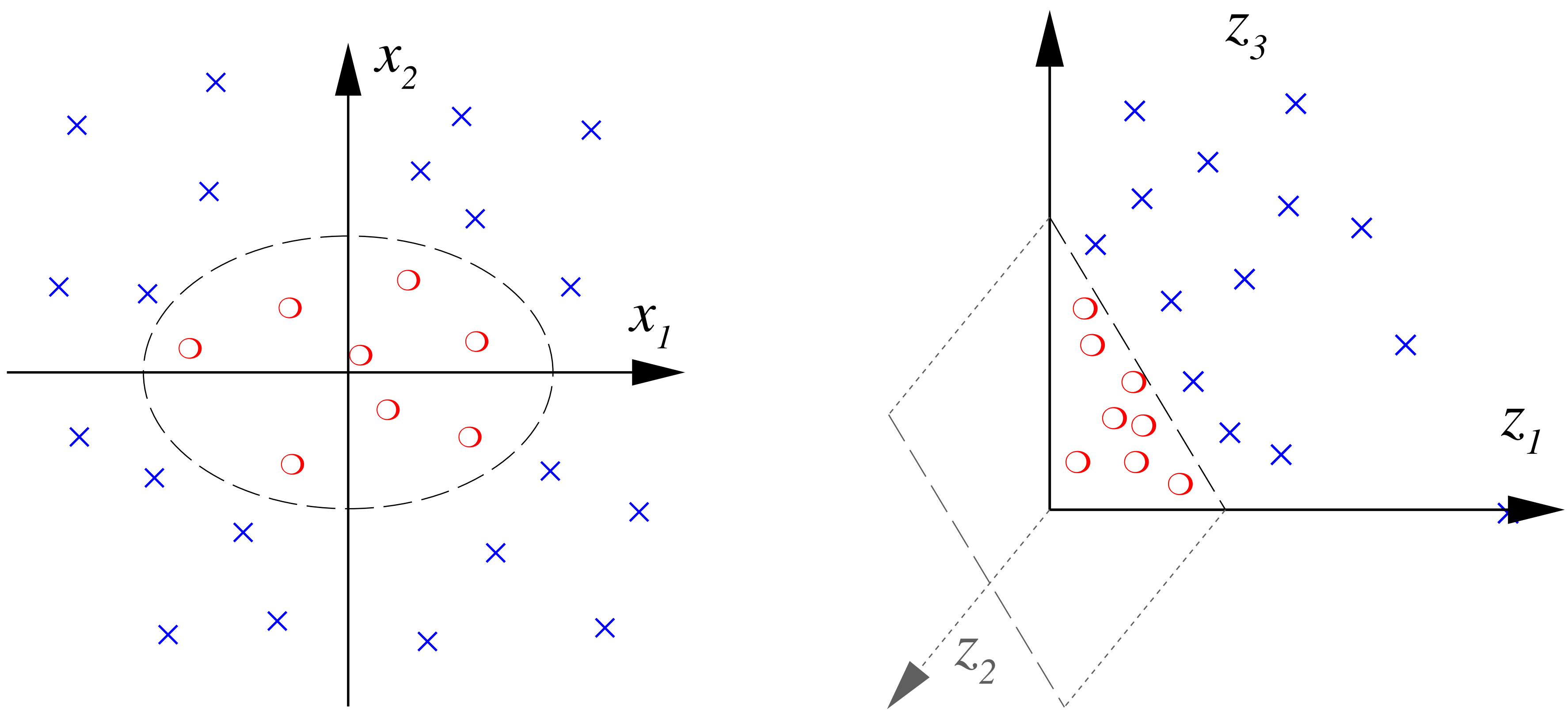
Support Vector Machine (SVM)

Extending the perceptron idea: use a **linear classifier with margin** and a **non-linear feature transformation**.



Nonlinear Transformation in Kernel Space

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$$



Lagrangian Optimization Theory

Optimization under constraints (Primal Problem):

Given an optimization problem with domain $\Omega \subseteq \mathbb{R}^d$,

$$\begin{array}{rll} \text{minimize} & f(\mathbf{w}), & \mathbf{w} \in \Omega \\ \text{subject to} & g_i(\mathbf{w}) \leq 0, & i = 1, \ldots, k \\ & h_i(\mathbf{w}) = 0, & i = 1, \ldots, m \end{array}$$

The **generalized Lagrangian function** is defined as

$$L(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{w}) + \sum_{i=1}^{k} \alpha_i g_i(\mathbf{w}) + \sum_{i=1}^{m} \beta_i h_i(\mathbf{w})$$

Lagrangian Dual Problem (1797)

Definition (Langrangian Dual Problem):

The respective **Lagrangian dual problem** is given by

$$\begin{array}{rl} \text{maximize} & \theta(\boldsymbol{\alpha}, \boldsymbol{\beta}), \\ \text{subject to} & \alpha_i \geq 0, \quad i = 1, \ldots, k \end{array}$$

$$\text{where} \qquad \theta(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \inf_{\mathbf{w} \in \Omega} L(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})$$

The value of the objective function at the optimal solution is called the **value of the problem**.

The **difference** between the values of the primal and the dual problems is known as the *duality gap*.

Upper Bound on Dual Costs

Theorem: Let $\mathbf{w} \in \Omega$ be a feasible solution of the primal problem of the previous definition and $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ a feasible solution of the respective dual problem. Then $f(\mathbf{w}) \geq \theta(\boldsymbol{\alpha}, \boldsymbol{\beta})$.

Proof:

$$\begin{aligned}
\theta(\boldsymbol{\alpha}, \boldsymbol{\beta}) &= \inf_{\mathbf{u} \in \Omega} L(\mathbf{u}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \\
&\leq L(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \\
&= f(\mathbf{w}) + \sum_{i=1}^{k} \underbrace{\alpha_i}_{\geq 0} \underbrace{g_i(\mathbf{w})}_{\leq 0} + \sum_{j=1}^{m} \beta_j \underbrace{h_j(\mathbf{w})}_{=0} \quad \leq f(\mathbf{w})
\end{aligned}$$

The feasibility of $\mathbf{w}$ implies $g_i(\mathbf{w}) \leq 0$ and $h_i(\mathbf{w}) = 0$, while the feasibility of $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ implies $\alpha_i \geq 0$.

Duality Gap

Corollary: The value of the dual problem is upper bounded by the value of the primal problem,

$$\sup \{\theta(\boldsymbol{\alpha}, \boldsymbol{\beta}) : \boldsymbol{\alpha} \geq 0\} \quad \leq \quad \inf \{f(\mathbf{w}) : \mathbf{g}(\mathbf{w}) \leq 0, \mathbf{h}(\mathbf{w}) = 0\}$$

Theorem: The triple $(\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ is a saddle point of the Lagrangian function for the primal problem, if and only if its components are optimal solutions of the primal and dual problems and if there is **no duality gap**, i.e., the primal and dual problems having the value

$$f(\mathbf{w}^*) = \theta(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$$

Strong Duality

Theorem: Given an optimization problem with convex objective function f and convex domain $\Omega \subseteq \mathbb{R}^d$,

$$\begin{array}{rll} \text{minimize} & f(\mathbf{w}), & \mathbf{w} \in \Omega \\ \text{subject to} & g_i(\mathbf{w}) \leq 0, & i = 1, \ldots, k \\ & h_i(\mathbf{w}) = 0, & i = 1, \ldots, m \end{array}$$

where the g_i and h_i are affine functions, that is

$$\mathbf{h}(\mathbf{w}) = \mathbf{A}\mathbf{w} - \mathbf{b},$$

for some matrix $\mathbf{A}$ and vector $\mathbf{b}$, then *the duality gap is zero*.

(This case applies to SVMs!)

Remark: If the functions $g_i(\mathbf{w})$ are convex then strong duality holds provided some *constraint qualifications* are fulfilled (e.g. Slater condition).

Kuhn-Tucker Conditions (1951)

Theorem: Given an optimization problem with convex domain $\Omega \subseteq \mathbb{R}^d$,

$$\begin{array}{rll} \text{minimize} & f(\mathbf{w}), & \mathbf{w} \in \Omega \\ \text{subject to} & g_i(\mathbf{w}) \leq 0, & i = 1, \ldots, k \\ & h_i(\mathbf{w}) = 0, & i = 1, \ldots, m \end{array}$$

with $f \in C^1$ convex and g_i, h_i affine, necessary and sufficient conditions for a normal point $\mathbf{w}^*$ to be an optimum are the existence of $\boldsymbol{\alpha}^*$, $\boldsymbol{\beta}^*$ such that

$$\frac{\partial L(\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \mathbf{w}} = 0 \qquad \frac{\partial L(\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \boldsymbol{\beta}} = 0$$

$$\alpha_i^* g_i(\mathbf{w}^*) = 0, \quad g_i(\mathbf{w}^*) \leq 0, \quad \alpha_i^* \geq 0, \quad i = 1, \ldots, k$$

Support Vector Machines (SVM)

Idea: linear classifier with margin and feature transformation.

Transformation from original feature space to nonlinear feature space.

$$\mathbf{y}_i = \phi(\mathbf{x}_i) \quad \text{e.g. Polynomial, Radial Basis Function, ...}$$

$$\phi : \mathbb{R}^d \to \mathbb{R}^e \text{ with } d \ll e$$

$$z_i = \begin{cases} +1 \\ -1 \end{cases} \text{if } \mathbf{x}_i \text{ in class } \begin{cases} y_1 \\ y_2 \end{cases}$$

Training vectors should be linearly separable after mapping!

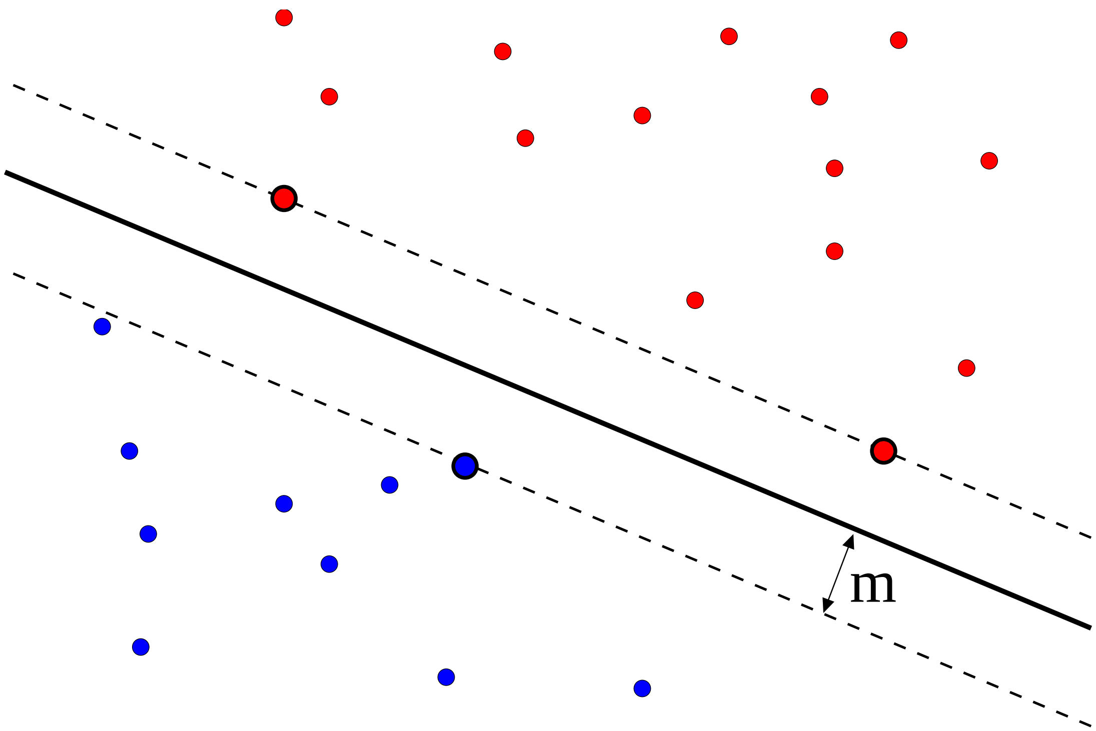
Linear discriminant function:

$$g(\mathbf{y}) = \mathbf{w}^\mathsf{T}\mathbf{y} + w_0$$

Support Vector Machine (SVM)

Find hyperplane that maximizes the **margin** m with

$$z_i\, g(\mathbf{y}_i) = z_i(\mathbf{w}^\mathsf{T}\mathbf{y} + w_0) \geq m \quad \text{for all } \mathbf{y}_i \in \mathcal{Y}$$



Vectors $\mathbf{y}_i$ with $z_i g(\mathbf{y}_i) = m$ are the **support vectors**.

Maximal Margin Classifier

Invariance: assume that the weight vector $\mathbf{w}$ is normalized ($\|\mathbf{w}\| = 1$) since a rescaling $(\mathbf{w}, w_0) \leftarrow (\lambda\mathbf{w}, \lambda w_0), m \leftarrow \lambda m$ does not change the problem.

Condition:
$$z_i = \begin{cases} +1 & \mathbf{w}^\mathsf{T}\mathbf{y}_i + w_0 \geq m \\ -1 & \mathbf{w}^\mathsf{T}\mathbf{y}_i + w_0 \leq -m \end{cases} \qquad \forall i$$

Objective: maximize margin m s.t. joint condition $z_i\,(\mathbf{w}^\mathsf{T}\mathbf{y}_i + w_0) \geq m$ is met.

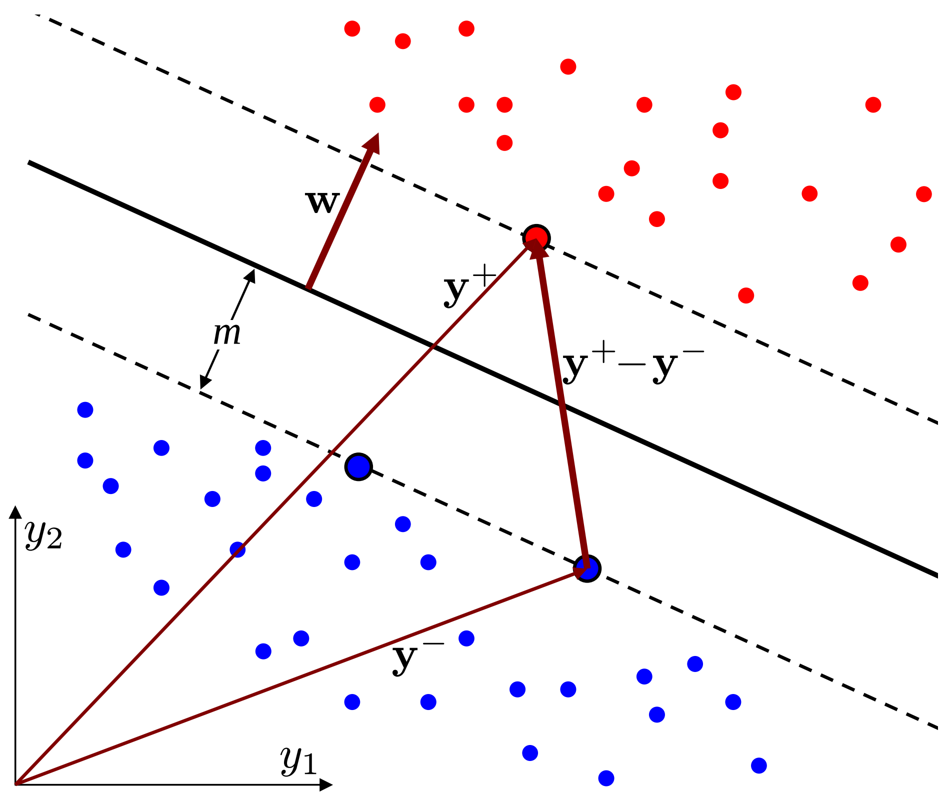
Learning problem: Find $\mathbf{w}$ with $||\mathbf{w}|| = 1$, such that the margin m is maximized.

$$\begin{aligned} &\text{maximize} \quad m \\ &\text{subject to} \quad \forall\,\mathbf{y}_i \in \mathcal{Y} : z_i(\mathbf{w}^\mathsf{T}\mathbf{y}_i + w_0) \geq m \end{aligned}$$

SVM Learning

What is the margin m?

Consider two points $\mathbf{y}^+, \mathbf{y}^-$ of class 1,2 which are located on both sides of the margin boundaries.



Transformation of objective:

rescaling $\mathbf{w} \leftarrow \frac{\mathbf{w}}{m}$, $w_0 \leftarrow \frac{w_0}{m}$ $\Rightarrow$

yields the constraints

$z_i(\mathbf{w}^\mathsf{T}\mathbf{y}_i + w_0) \geq 1$

Margin:

$$m = \frac{1}{2\|\mathbf{w}\|}(\mathbf{w}^\mathsf{T}\mathbf{y}^+ - \mathbf{w}^\mathsf{T}\mathbf{y}^-) = \frac{1}{\|\mathbf{w}\|}$$

$m = \frac{1}{\|\mathbf{w}\|}$ follows from inserting $\pm(\mathbf{w}^\mathsf{T}\mathbf{y}^\pm + w_0) = 1$

$\Rightarrow$ maximizing the margin corresponds to minimizing the norm $\|\mathbf{w}\|$ for margin $m = 1$.

SVM Lagrangian

Minimize $||\mathbf{w}||$ for a given margin $m = 1$

$$\begin{array}{rrcl} \text{minimize} & \mathcal{T}(\mathbf{w}) & = & \frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} \\ \text{subject to} & z_i(\mathbf{w}^{\mathsf{T}}\mathbf{y}_i + w_0) & \geq & 1 \end{array}$$

Generalized Lagrange Function:

$$L(\mathbf{w}, w_0, \boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} - \sum_{i=1}^{n} \alpha_i \left[z_i(\mathbf{w}^{\mathsf{T}}\mathbf{y}_i + w_0) - 1 \right]$$

Functional and geometric margin: The problem formulation with margin $m = 1$ is called the *functional margin* setting; The original formulation refers to the *geometric margin*.

Stationarity of Lagrangian

Extremality condition:

$$\frac{\partial L(\mathbf{w}, w_0, \boldsymbol{\alpha})}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i \le n} \alpha_i z_i \mathbf{y}_i = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{i \le n} \alpha_i z_i \mathbf{y}_i$$

$$\frac{\partial L(\mathbf{w}, w_0, \boldsymbol{\alpha})}{\partial w_0} = -\sum_{i \le n} \alpha_i z_i = 0$$

Resubstituting $\frac{\partial L}{\partial \mathbf{w}} = 0, \frac{\partial L}{\partial w_0} = 0$ into the Lagrangian function $L(\mathbf{w}, w_0, \boldsymbol{\alpha})$

$$\begin{aligned} L(\mathbf{w}, w_0, \boldsymbol{\alpha}) &= \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} - \sum_{i \le n} \alpha_i \left[z_i(\mathbf{w}^\mathsf{T}\mathbf{y}_i + w_0) - 1 \right] \\ &= \frac{1}{2}\sum_{i \le n}\sum_{j \le n} \alpha_i \alpha_j z_i z_j \mathbf{y}_i^\mathsf{T}\mathbf{y}_j - \sum_{i \le n}\sum_{j \le n} \alpha_i \alpha_j z_i z_j \mathbf{y}_i^\mathsf{T}\mathbf{y}_j + \sum_{i \le n} \alpha_i \\ &= \sum_{i \le n} \alpha_i - \frac{1}{2}\sum_{i \le n}\sum_{j \le n} \alpha_i \alpha_j z_i z_j \mathbf{y}_i^\mathsf{T}\mathbf{y}_j \qquad \text{(note the scalar product!)} \end{aligned}$$

Dual Problem

The **Dual Problem** for support vector learning is

maximize $W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} z_i z_j \alpha_i \alpha_j \mathbf{y}_i^{\mathsf{T}} \mathbf{y}_j$

subject to $\forall i \; \alpha_i \geq 0 \quad \wedge \quad \sum_{i=1}^{n} z_i \alpha_i = 0$

The optimal hyperplane $\mathbf{w}^*, w_0^*$ is given by

$$\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i^* z_i \mathbf{y}_i, \quad w_0^* = -\frac{1}{2} \left(\min_{i:z_i=1} \mathbf{w}^{*\mathsf{T}} \mathbf{y}_i + \max_{i:z_i=-1} \mathbf{w}^{*\mathsf{T}} \mathbf{y}_i \right)$$

where $\boldsymbol{\alpha}^*$ are the optimal Lagrange multipliers maximizing the Dual Problem.

Support Vectors

The **Kuhn-Tucker Conditions** for the maximal margin SVM are

$$
\begin{aligned}
\alpha_i^*(z_i g^*(\mathbf{y}_i) - 1) &= 0, \qquad i = 1, \ldots, n \\
\alpha_i^* &\geq 0, \qquad i = 1, \ldots, n \\
z_i g^*(\mathbf{y}_i) - 1 &\geq 0, \qquad i = 1, \ldots, n
\end{aligned}
$$

The first one is known as the **Kuhn-Tucker complementary condition**. The conditions imply

$$
\begin{aligned}
z_i g^*(\mathbf{y}_i) = 1 \;&\Rightarrow\; \alpha_i^* \geq 0 \qquad \text{Support Vectors (SV)} \\
z_i g^*(\mathbf{y}_i) \neq 1 \;&\Rightarrow\; \alpha_i^* = 0 \qquad \text{Non Support Vectors}
\end{aligned}
$$

Optimal Decision Function

Sparsity:

$$
\begin{aligned}
g^*(\mathbf{y}) &= \mathbf{w}^{*\mathsf{T}}\mathbf{y} + w_0^* = \sum_{i=1}^{n} z_i \alpha_i^* \mathbf{y}_i^{\mathsf{T}}\mathbf{y} + w_0^* \\
&= \sum_{i \in \mathsf{SV}} z_i \alpha_i^* \mathbf{y}_i^{\mathsf{T}}\mathbf{y} + w_0^*
\end{aligned}
$$

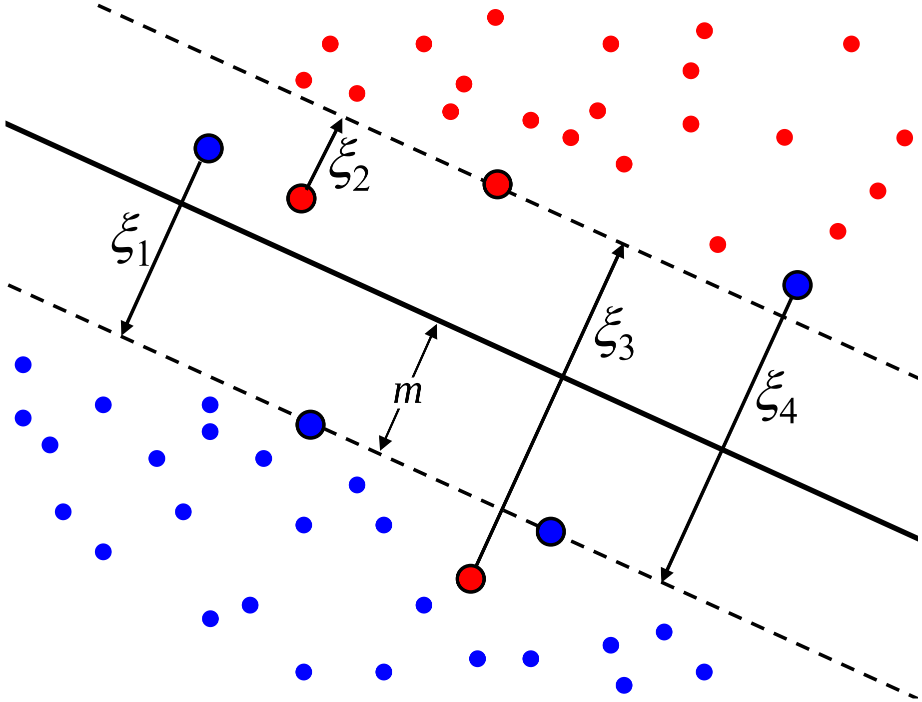
Remark: only few support vectors enter the sum to evaluate the decision function! $\Rightarrow$ efficiency and interpretability

Optimal margin: $\mathbf{w}^{\mathsf{T}}\mathbf{w} = \sum_{i \in \mathsf{SV}} \alpha_i^*$

Soft Margin SVM

For each trainings vector $\mathbf{y}_i \in \mathcal{Y}$ a **slack variable** ξ_i is introdu-ced to measure the violation of the margin constraint.

Find hyperplane that maximizes the margin $z_i g^*(\mathbf{y}_i) \geq m(1-\xi_i)$



Vectors $\mathbf{y}_i$ with $z_i g^*(\mathbf{y}_i) = m(1-\xi_i)$ are called **support vectors**.

Learning the Soft Margin SVM

Slack variables are penalized by L_1 norm.

$$\begin{aligned} \text{minimize} \qquad \mathcal{T}(\mathbf{w}, \boldsymbol{\xi}) &= \tfrac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} + C \textstyle\sum_{i=1}^{n} \xi_i \\ \text{subject to} \quad z_i(\mathbf{w}^{\mathsf{T}}\mathbf{y}_i + w_0) &\geq 1 - \xi_i \\ \xi_i &\geq 0 \end{aligned}$$

C controls the amount of constraint violations vs. margin maximization!

Lagrange function for soft margin SVM

$$\begin{aligned} L(\mathbf{w}, w_0, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &= \frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} + C \sum_{i=1}^{n} \xi_i \\ &- \sum_{i=1}^{n} \alpha_i \left[z_i(\mathbf{w}^{\mathsf{T}}\mathbf{y}_i + w_0) - 1 + \xi_i \right] - \sum_{i=1}^{n} \beta_i \xi_i \end{aligned}$$

Stationarity of Primal Problem

Differentiation:

$$\frac{\partial L(\mathbf{w}, w_0, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} \alpha_i z_i \mathbf{y}_i = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i z_i \mathbf{y}_i$$

$$\frac{\partial L(\mathbf{w}, w_0, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial \xi_i} = C - \alpha_i - \beta_i = 0 \quad \frac{\partial L(\mathbf{w}, w_0, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial w_0} = -\sum_{i=1}^{n} \alpha_i z_i = 0$$

Resubstituting into the Lagrangian function $L(\mathbf{w}, w_0, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ yields

$$\begin{aligned} L(\mathbf{w}, w_0, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &= \tfrac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + C \textstyle\sum_{i=1}^{n} \xi_i \\ &- \textstyle\sum_{i=1}^{n} \alpha_i \left[z_i(\mathbf{w}^\mathsf{T}\mathbf{y}_i + w_0) - 1 + \xi_i \right] - \sum_{i=1}^{n} \beta_i \xi_i \end{aligned}$$

$$
\begin{aligned}
L(\mathbf{w}, w_0, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &= \tfrac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j z_i z_j \mathbf{y}_i^{\mathsf{T}}\mathbf{y}_j + C\sum_{i=1}^{n}\xi_i \\
&\quad - \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j z_i z_j \mathbf{y}_i^{\mathsf{T}}\mathbf{y}_j \\
&\quad + \sum_{i=1}^{n}\alpha_i(1-\xi_i) - \sum_{i=1}^{n}\beta_i\xi_i \\
&= \sum_{i=1}^{n}\alpha_i - \tfrac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j z_i z_j \mathbf{y}_i^{\mathsf{T}}\mathbf{y}_j \\
&\quad + \sum_{i=1}^{n}(\underbrace{C-\alpha_i-\beta_i}_{=\frac{\partial L}{\partial \xi_i}=0})\xi_i \\
&= \sum_{i=1}^{n}\alpha_i - \tfrac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j z_i z_j \mathbf{y}_i^{\mathsf{T}}\mathbf{y}_j
\end{aligned}
$$

Constaints of the Dual Problem

The dual objective function is the same as for the maximal margin SVM. The only difference is the constraint

$$C - \alpha_i - \beta_i = 0$$

Together with $\beta_i \geq 0$ it implies

$$\alpha_i \leq C$$

The Kuhn-Tucker complementary conditions

$$\begin{aligned} \alpha_i(z_i(\mathbf{w}^\mathsf{T}\mathbf{y}_i + w_0) - 1 + \xi_i) &= 0, \qquad i = 1, \ldots, n \\ \xi_i(\alpha_i - C) &= 0, \qquad i = 1, \ldots, n \end{aligned}$$

imply that nonzero slack variables can only occur when $\alpha_i = C$.

Dual Problem of Soft Margin SVM

The **Dual Problem** for support vector learning is

$$\begin{array}{rl} \text{maximize} & W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} z_i z_j \alpha_i \alpha_j \mathbf{y}_i^\mathsf{T} \mathbf{y}_j \\ \text{subject to} & \sum_{j=1}^{n} z_j \alpha_j = 0 \ \wedge \ \forall i \ C \geq \alpha_i \geq 0 \end{array}$$

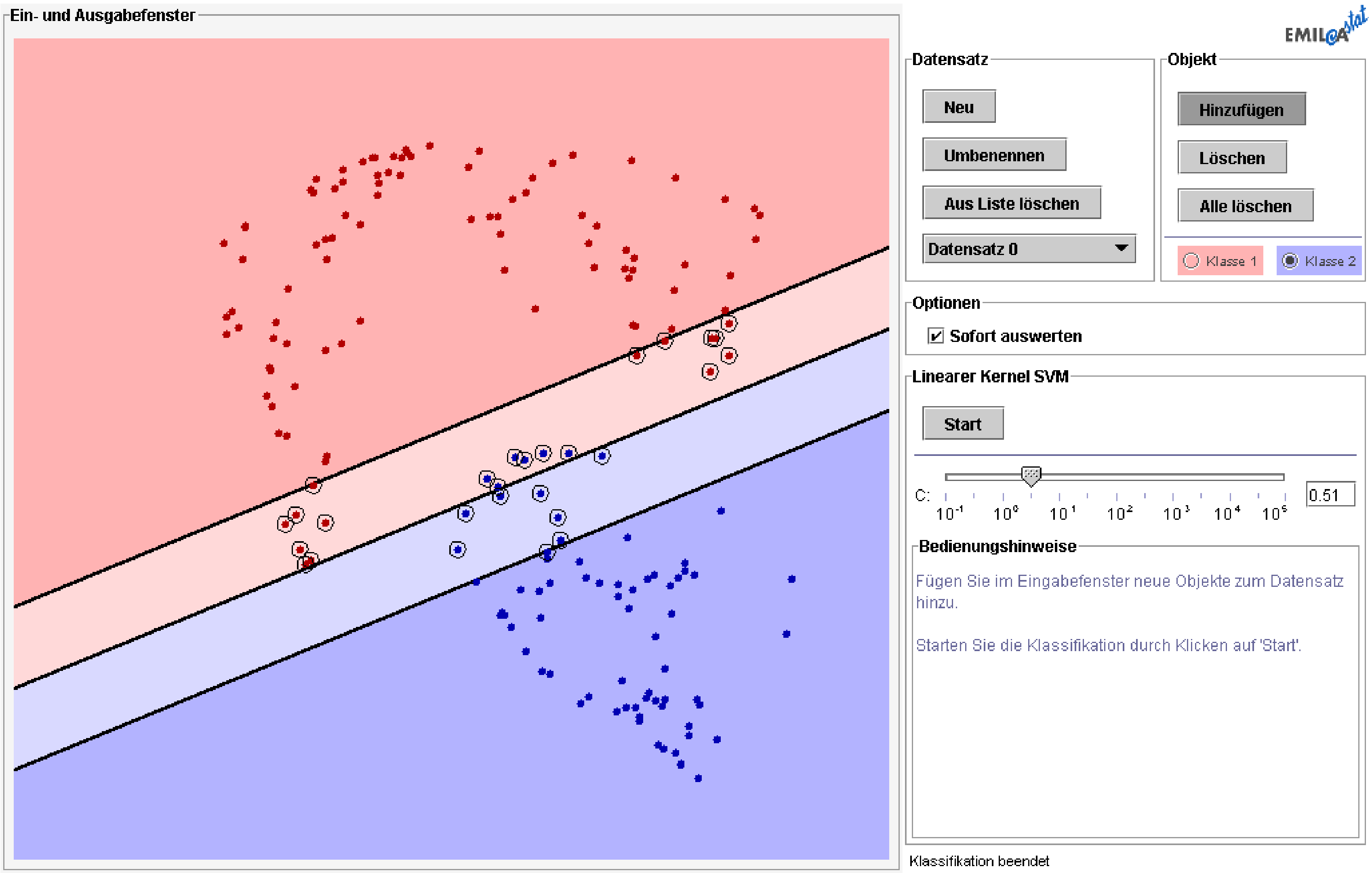
The optimal hyperplane $\mathbf{w}^*$ is given by

$$\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i^* z_i \mathbf{y}_i$$

where $\boldsymbol{\alpha}^*$ are the optimal Lagrange multipliers maximizing the Dual Problem.

$\alpha_i^* > 0$ holds only for **support vectors**.

Applet HTML Page



Linear Programming Support Vector Machines

Idea: Minimize an estimate of the number of positive multipliers $\sum_{i=1}^{n} \alpha_i$ which improves bounds on the generalization error.

The **Lagrangian** for the LP-SVM is

$$
\begin{aligned}
\text{minimize} \quad & W(\boldsymbol{\alpha}, \xi) = \sum_{i=1}^{n} \alpha_i + C \sum_{i=1}^{n} \xi_i \\
\text{subject to} \quad & z_i \left[\sum_{j=1}^{n} \alpha_j \mathbf{y}_i^{\mathsf{T}} \mathbf{y}_j + w_0 \right] \geq 1 - \xi_i, \\
& \alpha_i \geq 0,\ \xi_i \geq 0,\ 1 \leq i \leq n
\end{aligned}
$$

Advantage: efficient LP solver can be used.

Disadvantage: theory is not as well understood as for standard SVMs.

Non–Linear SVMs

Feature extraction by non linear transformation $\mathbf{y} = \phi(\mathbf{x})$

Problem:

$$\mathbf{y}_i^{\mathsf{T}} \mathbf{y}_j = \phi^{\mathsf{T}}(\mathbf{x}_i)\phi(\mathbf{x}_j)$$

is the inner product in a high dimensional space.

A **kernel function** is defined by

$$\forall \mathbf{x}, \mathbf{z} \in \Omega : \quad K(\mathbf{x}, \mathbf{z}) = \phi^{\mathsf{T}}(\mathbf{x})\phi(\mathbf{z})$$

Using the kernel function the discriminant function becomes

$$g(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i z_i \underbrace{K(\mathbf{x}_i, \mathbf{x})}_{\text{replaces } \mathbf{y}_i^{\mathsf{T}}\mathbf{y}}$$

Characterization of Kernels

For any symmetric matrix $K(\mathbf{x}_i, \mathbf{x}_j)|_{i,j=1}^{n}$ (Gram matrix) there exists an eigenvector decomposition

$$K = V\Lambda V^{\mathsf{T}}.$$

V: orthogonal matrix of eigenvectors $(v_{ti})|_{i=1}^{n}$
Λ: diagonal matrix of eigenvalues λ_t

Assume all eigenvalues are nonnegative and consider mapping

$$\phi : \mathbf{x}_i \to \left(\sqrt{\lambda_t} v_{ti}\right)_{t=1}^{n} \in \mathbb{R}^n, i = 1, \ldots, n$$

Then it follows

$$\phi^{\mathsf{T}}(\mathbf{x}_i)\phi(\mathbf{x}_j) = \sum_{t=1}^{n} \lambda_t v_{ti} v_{tj} = \left(V\Lambda V^{\mathsf{T}}\right)_{ij} = K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$$

Positivity of Kernels

Theorem: Let Ω be a finite input space with $K(\mathbf{x}, \mathbf{z})$ a symmetric function on Ω. Then $K(\mathbf{x}, \mathbf{z})$ is a kernel function if and only if the matrix

$$K = (K(\mathbf{x}_i, \mathbf{x}_j))_{i,j=1}^{n}$$

is *positive semi-definite* (has only non-negative eigenvalues).

Extension to infinite dimensional Hilbert Spaces:

$$< \phi(\mathbf{x}), \phi(\mathbf{z}) >= \sum_{i=1}^{\infty} \lambda_i \phi_i(\mathbf{x}) \phi_i(\mathbf{z})$$

Mercer's Theorem

Theorem (Mercer): Let Ω be a compact subset of $\mathbb{R}^n$. Suppose K is a continuous symmetric function such that the integral operator $T_K : L_2(X) \to L_2(X)$,

$$(T_K f)(\cdot) = \int_\Omega K(\cdot, \mathbf{x}) f(\mathbf{x}) d\mathbf{x},$$

is positive, that is $\int_{\Omega \times \Omega} K(\mathbf{x}, \mathbf{z}) f(\mathbf{x}) f(\mathbf{z}) d\mathbf{x} d\mathbf{z} > 0 \quad \forall f \in L_2(\Omega)$

Then we can expand $K(\mathbf{x}, \mathbf{z})$ in a uniformly convergent series in terms of T_K's eigen-functions $\phi_j \in L_2(\Omega)$, with $||\phi_j||_{L_2} = 1$ and $\lambda_j > 0$.

Possible Kernels

Remark: Each kernel function, that hold Mercer's conditions describes an inner product in a high dimensional space. The kernel function replaces the inner product.

Possible Kernels:

$$a) \quad K(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{||\mathbf{x}-\mathbf{z}||^2}{2\sigma^2}\right) \qquad \text{(RBF Kernel)}$$

$$b) \quad K(\mathbf{x}, \mathbf{z}) = \tanh \kappa \mathbf{x}\mathbf{z} - b \qquad \text{(Sigmoid Kernel)}$$

$$c) \quad K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}\mathbf{z})^d \qquad \text{(Polynomial Kernel)}$$

$$K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}\mathbf{z} + 1)^d$$

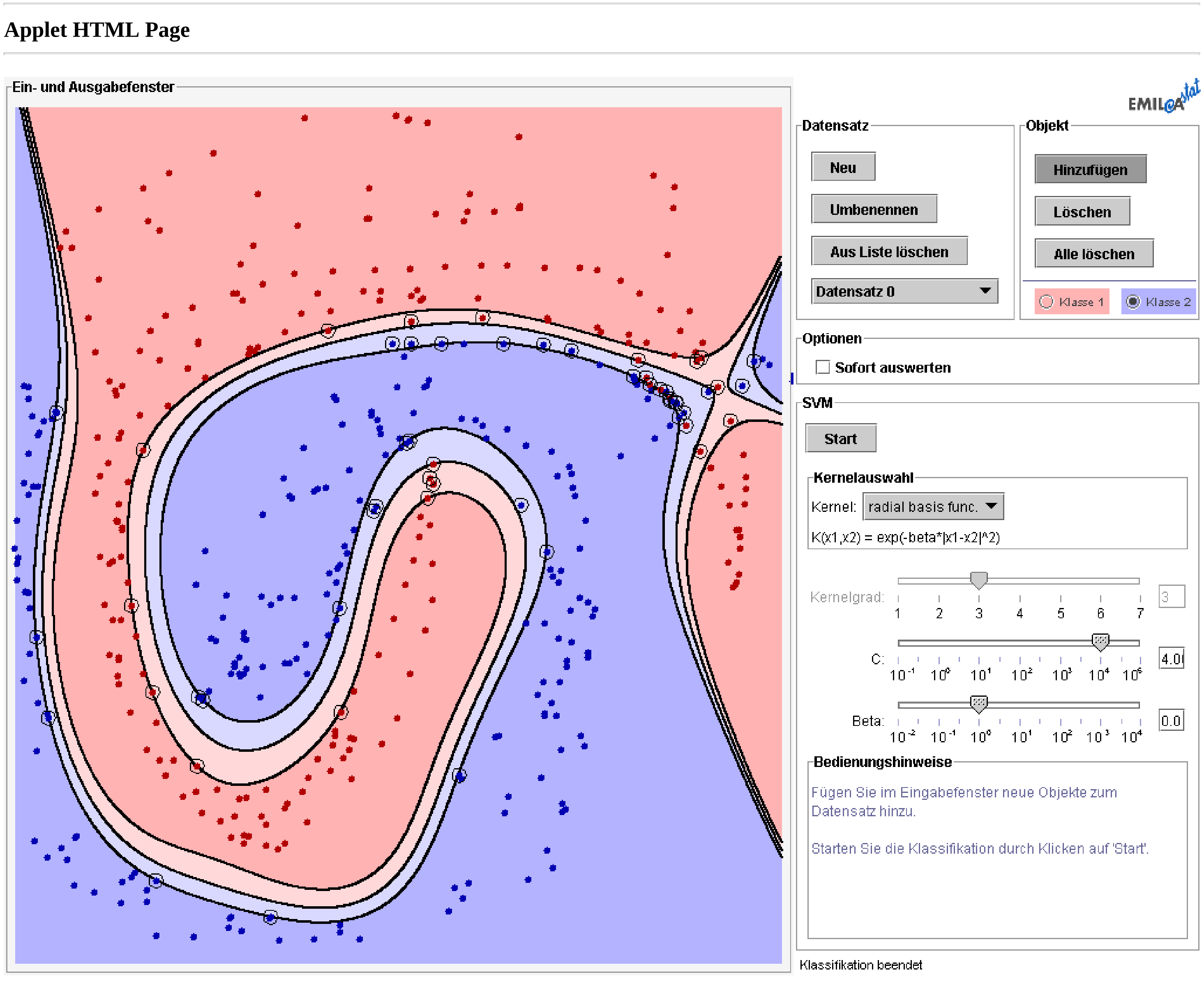
$d) \quad K(\mathbf{x}, \mathbf{z})$: string kernels for sequences

Kernel Engineering

Kernel composition rules: Let K_1, K_2 be kernels over $\Omega \times \Omega, \Omega \subseteq \mathbb{R}^d$, $a \in \mathbb{R}^+, f(.)$ a real-vealued function $\phi : \Omega \to \mathbb{R}^e$ with K_3 a kernel over $\mathbb{R}^e \times \mathbb{R}^e$.

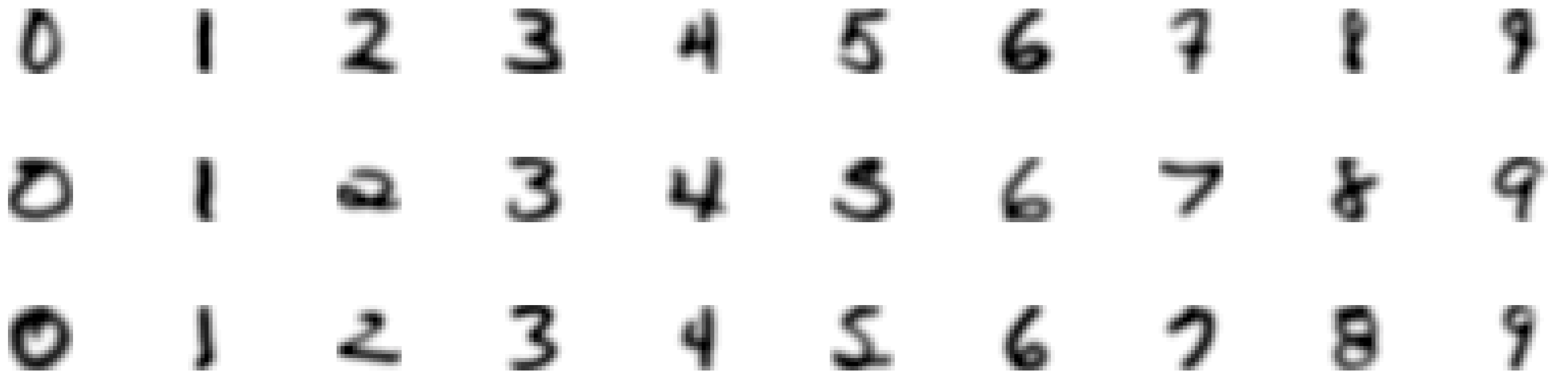
Then the following functions are kernels:

1. $K(\mathbf{x}, \mathbf{z}) = K_1(\mathbf{x}, \mathbf{z}) + K_2(\mathbf{x}, \mathbf{z})$,
2. $K(\mathbf{x}, \mathbf{z}) = aK_1(\mathbf{x}, \mathbf{z})$,
3. $K(\mathbf{x}, \mathbf{z}) = K_1(\mathbf{x}, \mathbf{z})K_2(\mathbf{x}, \mathbf{z})$,
4. $K(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})f(\mathbf{z})$,
5. $K(\mathbf{x}, \mathbf{z}) = K_3(\phi(\mathbf{x}), \phi(\mathbf{z}))$,
6. $K(\mathbf{x}, \mathbf{z}) = p(K_1(\mathbf{x}, \mathbf{z}))$, ($p(x)$ is a polynomial with positive coefficients)
7. $K(\mathbf{x}, \mathbf{z}) = \exp(K_1(\mathbf{x}, \mathbf{z}))$,


Applet HTML Page
Ein- und Ausgabefenster
EMILeA-stat
Datensatz
Neu
Umbenennen
Aus Liste löschen
Datensatz 0
Objekt
Hinzufügen
Löschen
Alle löschen
Klasse 1
Klasse 2
Optionen
Sofort auswerten
SVM
Start
Kernelauswahl
Kernel: radial basis func.
K(x1,x2) = exp(-beta*|x1-x2|^2)
Kernelgrad:
C:
Beta:
Bedienungshinweise
Fügen Sie im Eingabefenster neue Objekte zum Datensatz hinzu.
Starten Sie die Klassifikation durch Klicken auf 'Start'.
Klassifikation beendet

Example: Hand Written Digit Recognition

- 7291 training images und 2007 test images (16x16 pixel, 256 gray values)



Classification method	test error
human classification	2.7 %
perceptron	5.9 %
support vector machines	4.0 %

SVMs for Secondary Structure Prediction

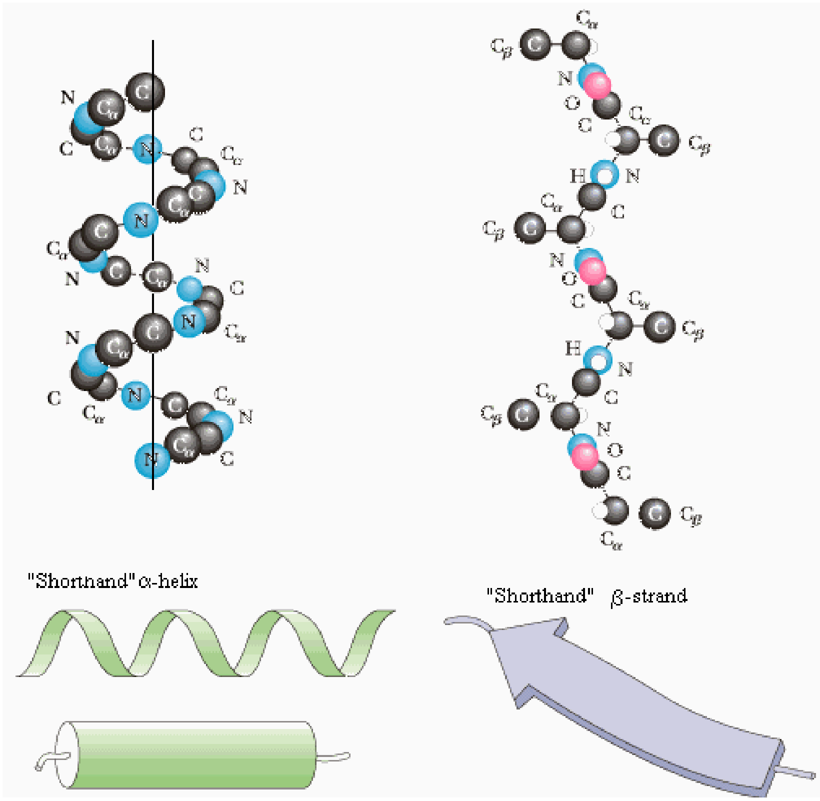
Proteins are represented in “zeroth order” by the percentage of amino-acids in the polypeptide chain; $\rightsquigarrow$ “vectorial” representation in $\mathbb{R}^{20}$

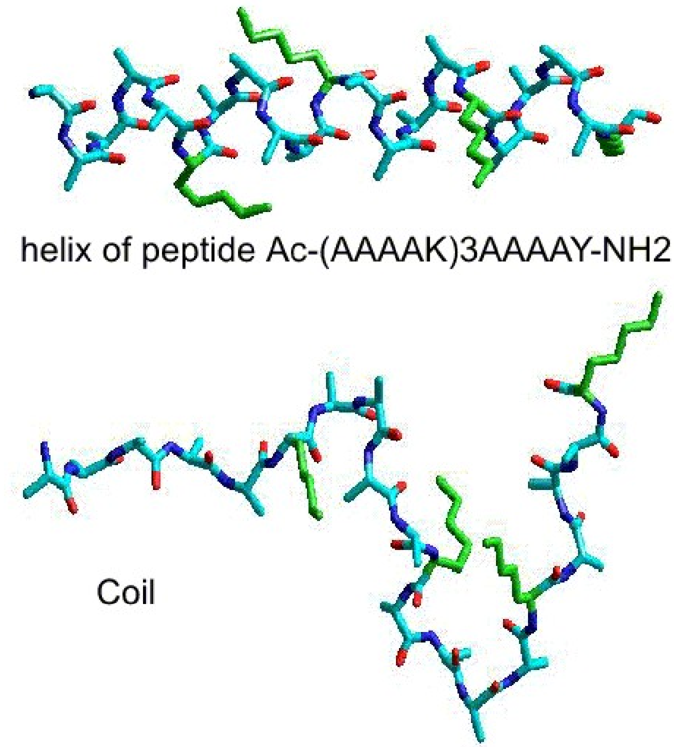
Protein structure problem: **sequence** as primary structure, **local motives** as secondary structure, **protein folds** as ternary structure.

SVM classification typically use the *RBF* kernel

$$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\gamma \|\mathbf{x} - \mathbf{y}\|^2\right)$$

Secondary structure prediction as a multiclass problem: Detect classes *helix (H)*, *sheet (E)* and *coil (C)*





Accuracy measure: $Q_3 = \%$ of correct 3-state symbols, i.e.

$$Q_3 = \frac{\text{\#correctly predicted residues}}{\text{total \# of residues}} \cdot 100$$

Practical Problem: How to apply SVMs for $k > 2$ classes?

Linear Discriminants and the Multicategory Case

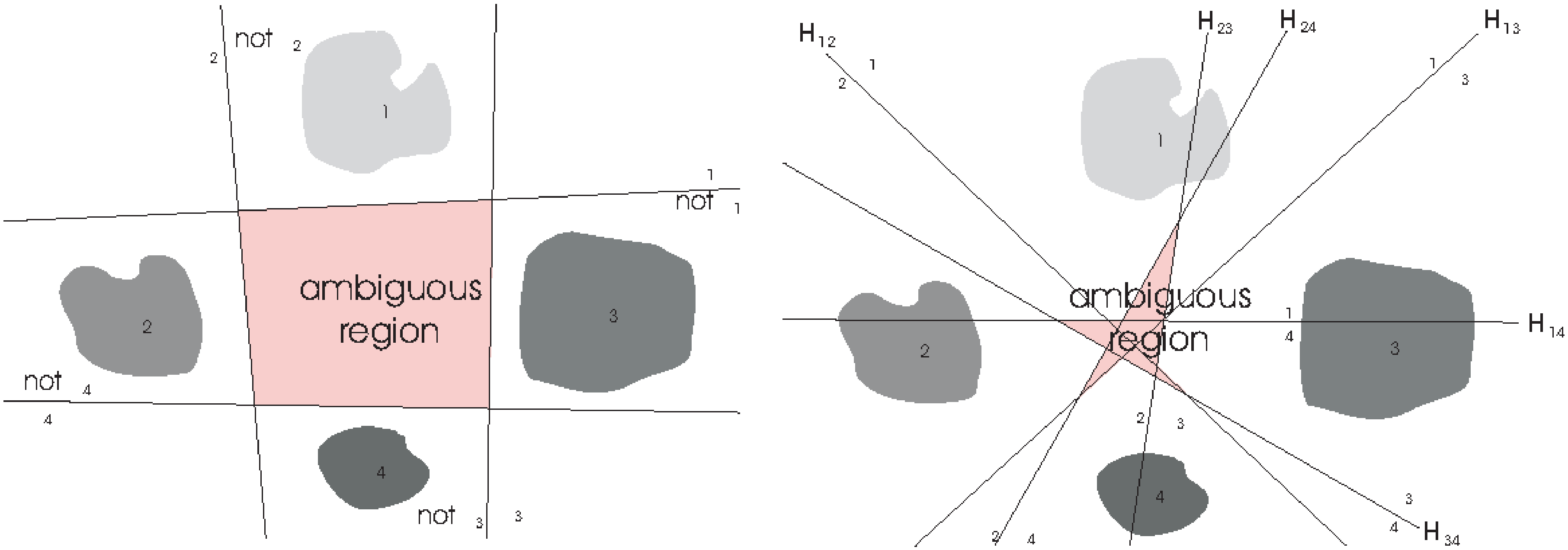
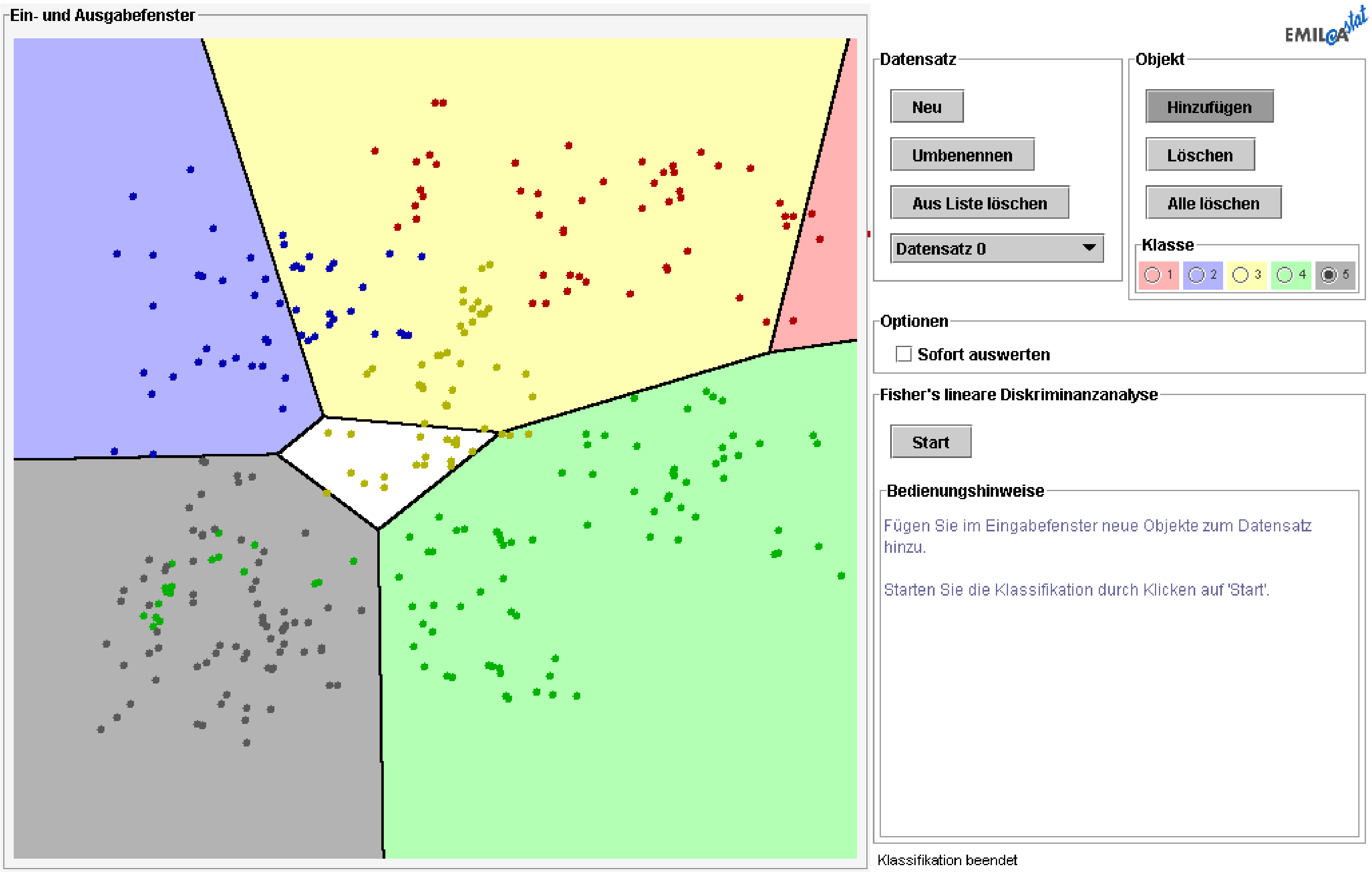


FIGURE 5.3. Linear decision boundaries for a four-class problem. The Top figure shows ω_i/not ω_i dichotomies while the bottom figure shows ω_i/ω_j dichotomies and the corresponding decision boundaries H_{ij}. The pink regions have ambiguous category assignments. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*.

Idea: it is often preferable to reformulate the multiclass problem as $(k-1)$ "*class* α *– not class* α" dichotomies or $k(k-1)/2$ "*class* α *or* β" dichotomies.

Problem: some areas in feature space are ambiguously classified.

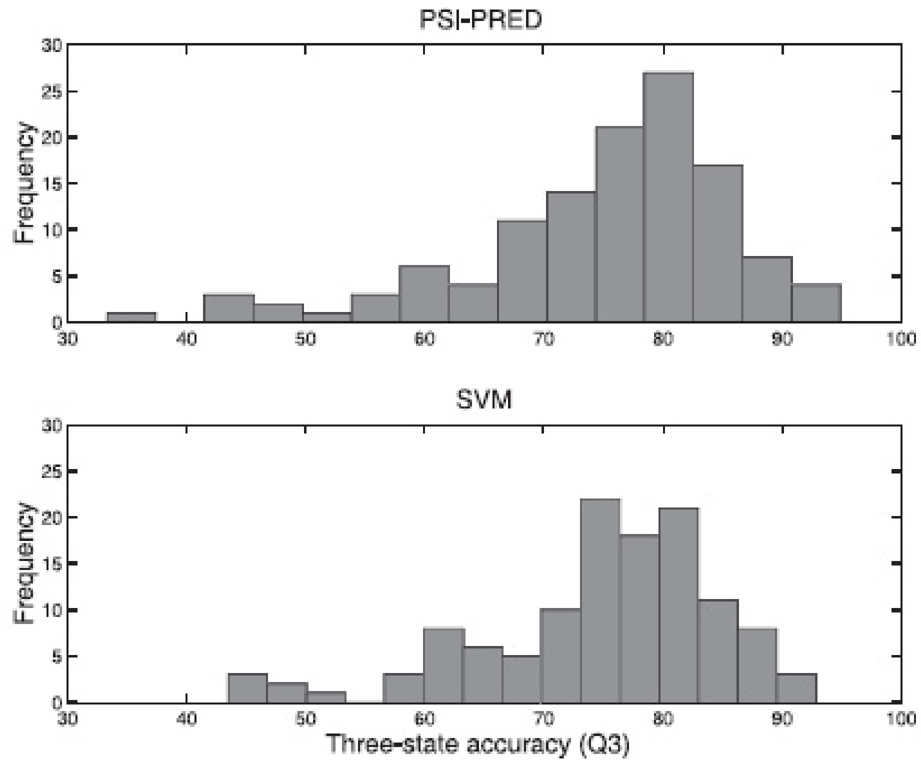
Applet HTML Page



Generated by NetBeans IDE

Experimental Results

- *PHD* (by B. Rost *et al.*, Neural Network based approach) – 72-74% Q_3
- *Psi-pred* (by D. T. Jones *et al.*, Neural Network based approach) – 76-78% Q_3
- The extensive study by *Ward* et al. (Bioinformatics, 2003) with different SVM realization reports results 73-77% Q_3
- Two-layer classification strategy with position-specific scoring scheme (*Guo* et al., Proteins, 2004)). Accuracy ranges from 78% – 80%.



Histogram of Q_3 scores for 121 test proteins (Ward *et al.*, Bioinformatics 19:13, 2003)

Machine Learning on Audio Data

Project with the company Phonak (Stäfa), producer of hearing aids.

Task: Given an accoustic environment, find appropriate control settings for the hearing aid:

- Speech understanding in silent and noisy environments
- Natural hearing of music and sounds in nature
- Comfortable setting for noisy environments

Classification of Audio Data

Current setting: Four sound classes are defined corresponding to the basic hearing goals:

$\longrightarrow$ *Speech*
$\longrightarrow$ *Speech in Noise*
$\longrightarrow$ *Music*
$\longrightarrow$ *Noise*

Goal: Let the hearing instrument autonomously decide which environment you are in!

Question: Are the four sound classes supported by sound statistics?

Features from Audio Data

Feature set: Common features are

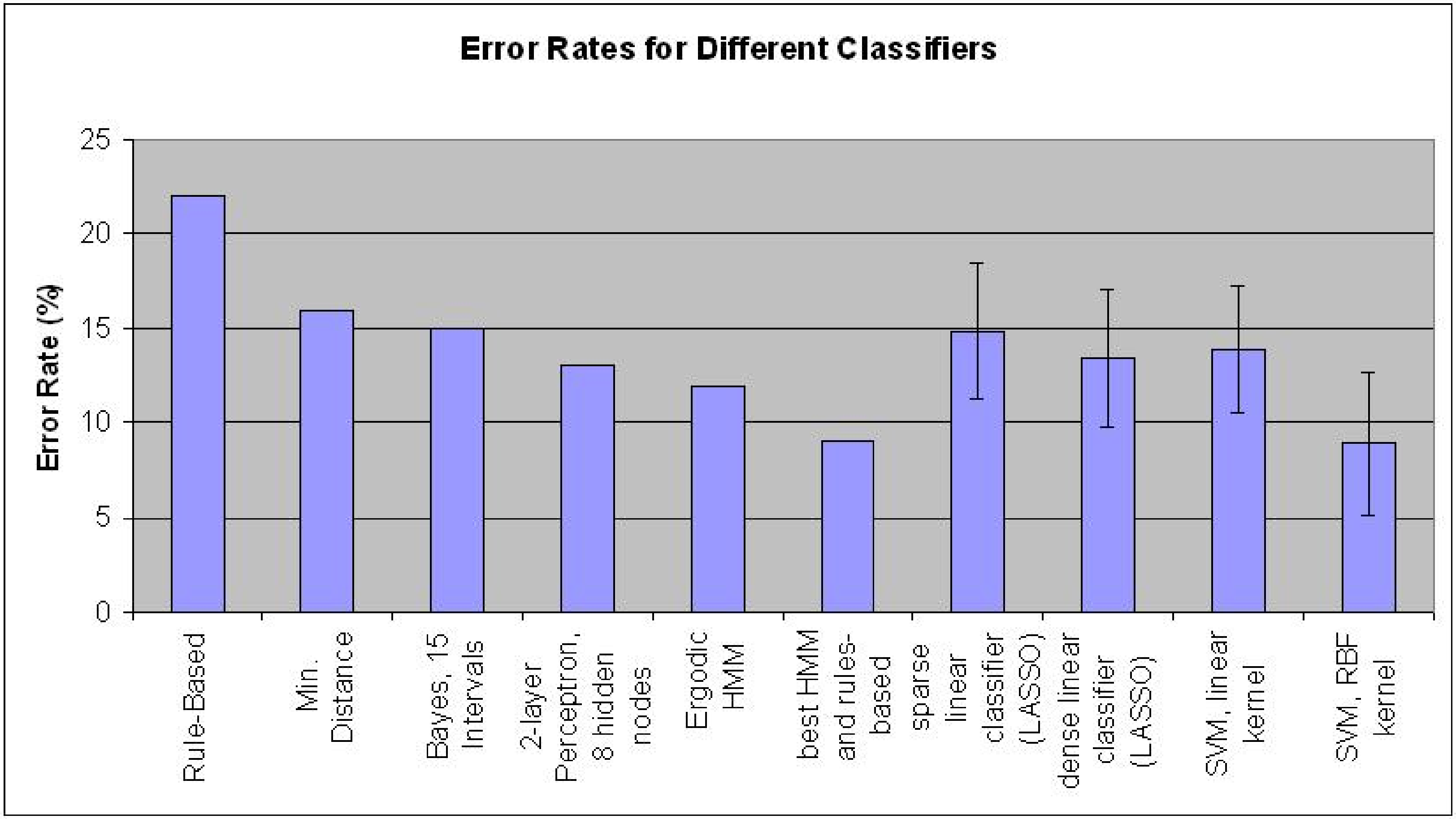
- distribution of the spectrum
- tonality
- rhythm
- estimated signal to noise ratio (SNR)
- ... and others

Strong computational constraints in the hearing aid!

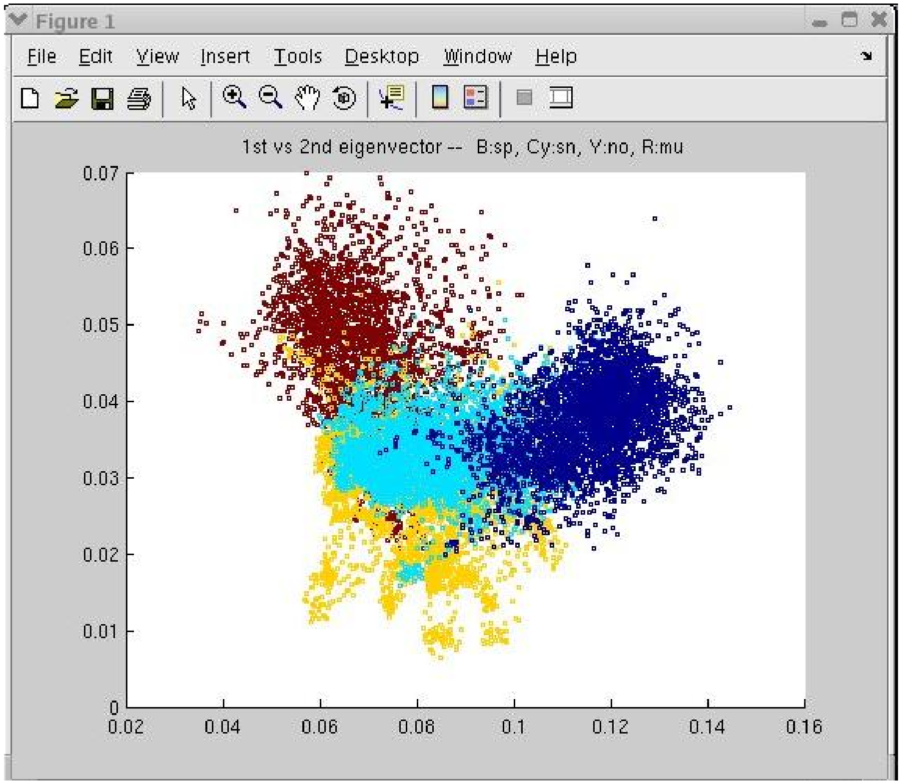
- Very little computational power and memory is available.
- Delay must not exceed a few ten miliseconds

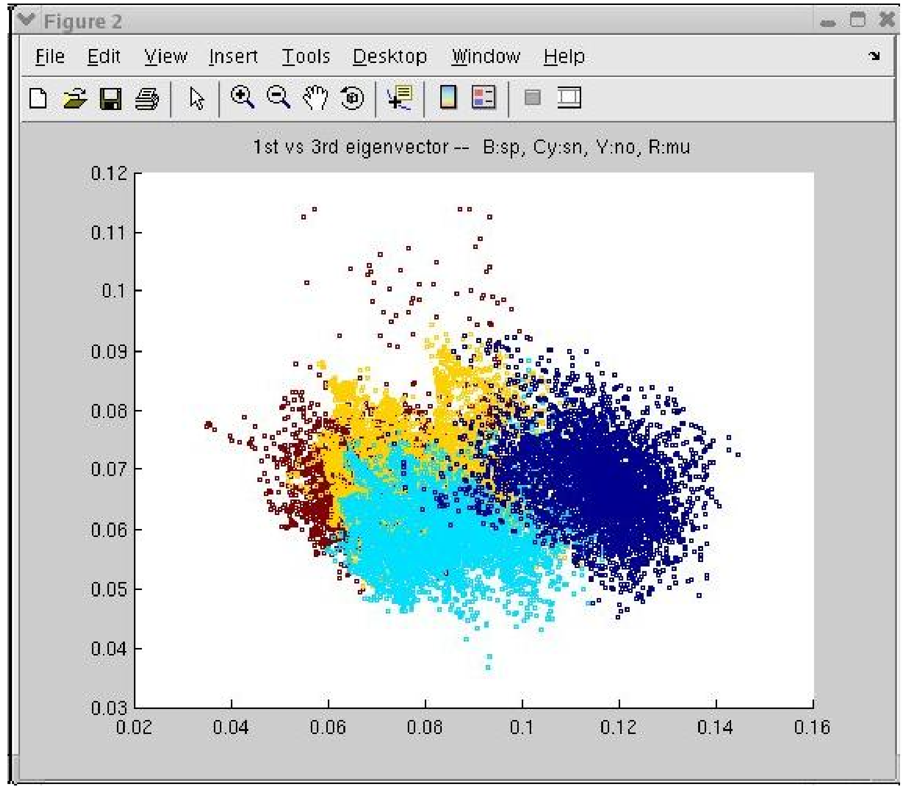
$\longrightarrow$ Complex features can only be approximated.

Classification Quality for different Classifiers



Linear Discriminant Analysis

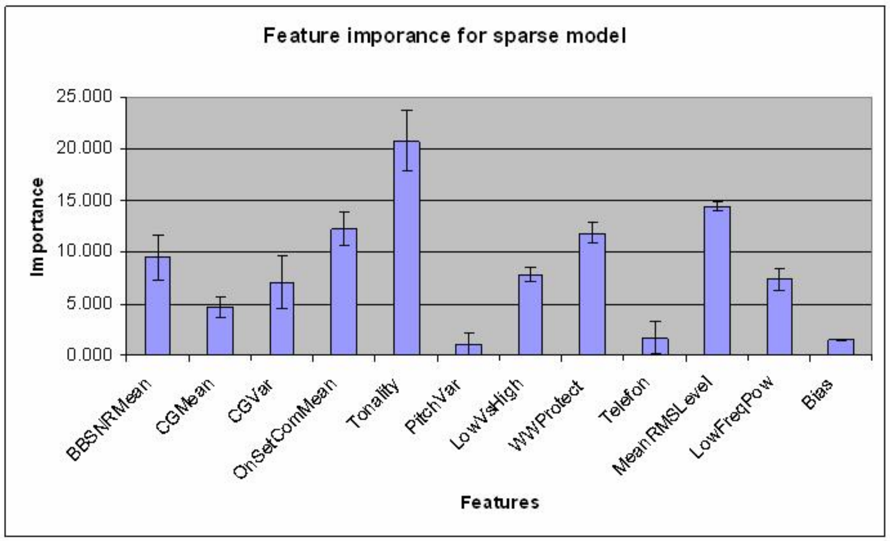


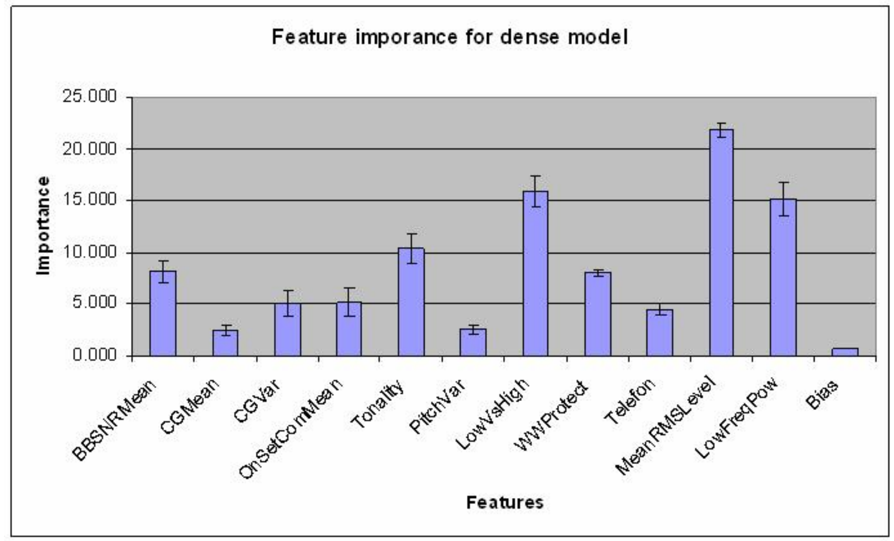


- Speech and most music files can be clearly separated.
- Speech in noise and noise are substantially overlapping.

Feature importance

Relative feature importance for a sparse and a dense linear model:





- All of the currently used features are used ...
- ... but not all features have the same importance.

Machine Learning: Topic Chart

- **Core problems of pattern recognition**
- **Bayesian decision theory**
- **Perceptrons and Support vector machines**
- ***Data clustering***
- **Dimension reduction**

Supervised vs. Unsupervised learning

Training data: A sample from the data source with the correct classification/regression solution already assigned.

Supervised learning = Learning based on training data.

Two steps:

1. Training step: Learn classifier/regressor from training data.
2. Prediction step: Assign class labels/functional values to test data.

Perceptron, LDA, *SVMs*, linear/ridge/kernel ridge regression are all supervised methods.

Unsupervised learning: Learning without training data.

Unsupervised learning

Examples:

- Data clustering. (Some authors do not distinguish between clustering and unsupervised learning.)
- Dimension reduction techniques.

Data clustering: Divide input data into groups of similar points.
$\rightarrow$ Roughly the unsupervised counterpart to classification.

Note the difference:

- Supervised case: Fit model to each class of training points, then use models to classify test points.
- Clustering: Simultaneous inference of group structure and model.

Grouping or Clustering: the k-Means Problem

Given are d-dimensional sample vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$

Define ...

- ... assignment vector $c \in \{1, \ldots, k\}^n$
- ... prototypes $\mathbf{y}_\nu \in \mathcal{Y} \subset \mathbb{R}^d$ ($\nu \in \{1, \ldots, k\}$)

Problem: Find c and $\mathbf{y}_\nu$ such that the clustering costs are minimized ($c_i := c(\mathbf{x}_i)$)

$$R^{\text{km}}(c, \mathcal{Y}) = \sum_{i=1}^{n} ||\mathbf{x}_i - \mathbf{y}_{c_i}||^2$$

Mixed combinatorial and continuous optimization problem

k-Means Algorithm

1. **Choose** k sample objects randomly as prototypes, i.e., select $\mathcal{Y} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$

2. **Iterate**:

   - Keep prototypes $\mathbf{y}_{c_i}$ fixed and assign sample vectors $\mathbf{x}_i$ to nearest prototype

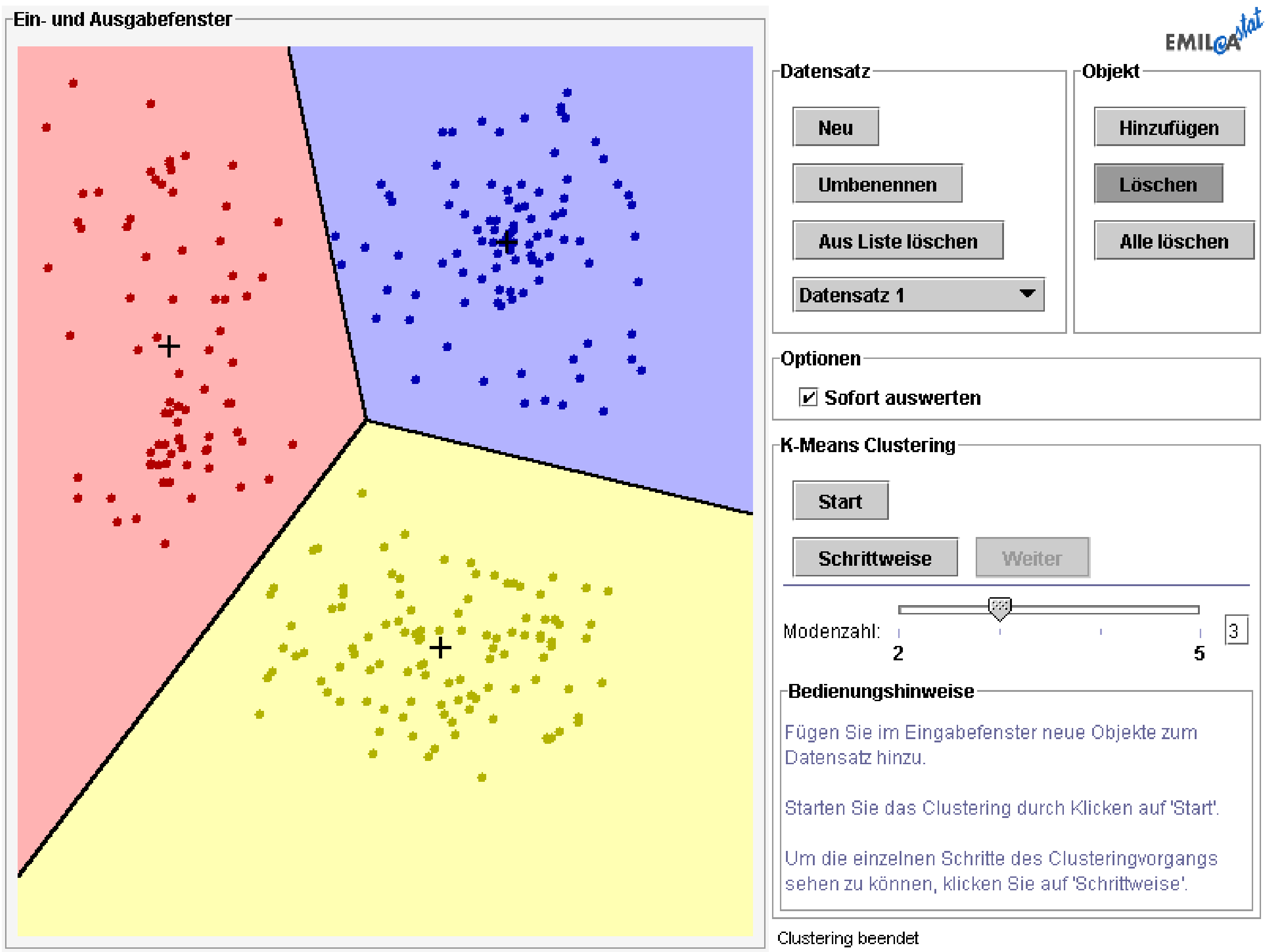
   $$c_i = \arg \min_{\nu \in \{1,\ldots,k\}} ||\mathbf{x}_i - \mathbf{y}_{c_i}||^2$$

   - Keep assignments c_i fixed and estimate prototypes

   $$\mathbf{y}_\nu = \frac{1}{n_\nu} \sum_{i:c_i=\nu} \mathbf{x}_i \quad \text{with} \quad n_\nu = |\{i : c_i = \nu\}|$$

Clustering of Vector Data

Applet HTML Page



Generated by NetBeans IDE

Mixture models

Def.: A *finite mixture model* is a probability density of the form

$$p(x) = \sum_{j=1}^{l} c_j p_j(x)$$

where the p_j are probability densities on a common domain Ω, $c_j \geq 0$ constants and $\sum_j c_j = 1$.

Remarks:

- p is a density on Ω.
- If all components are parametric models, then so is p.
- Most common: Gaussian mixture, $p_j(x) := g(x|\mu_j, \sigma_j)$.

Mixture models: Interpretation

Recall: Addition on probabilities $\leftrightarrow$ logical OR.

Represented data source:

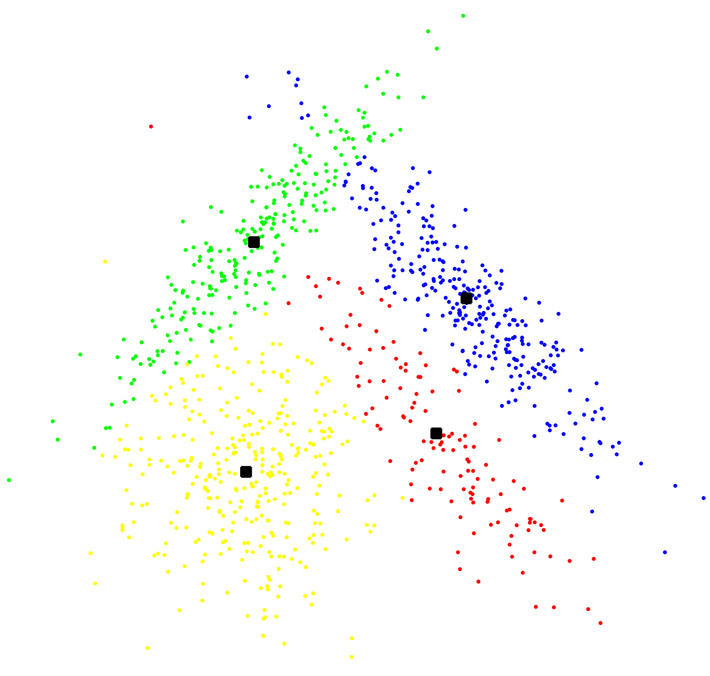
- Source = set of component sources (modeled by the p_j)
- Each data value is drawn from exactly one component source.
- c_j: Probability of draw from p_j.

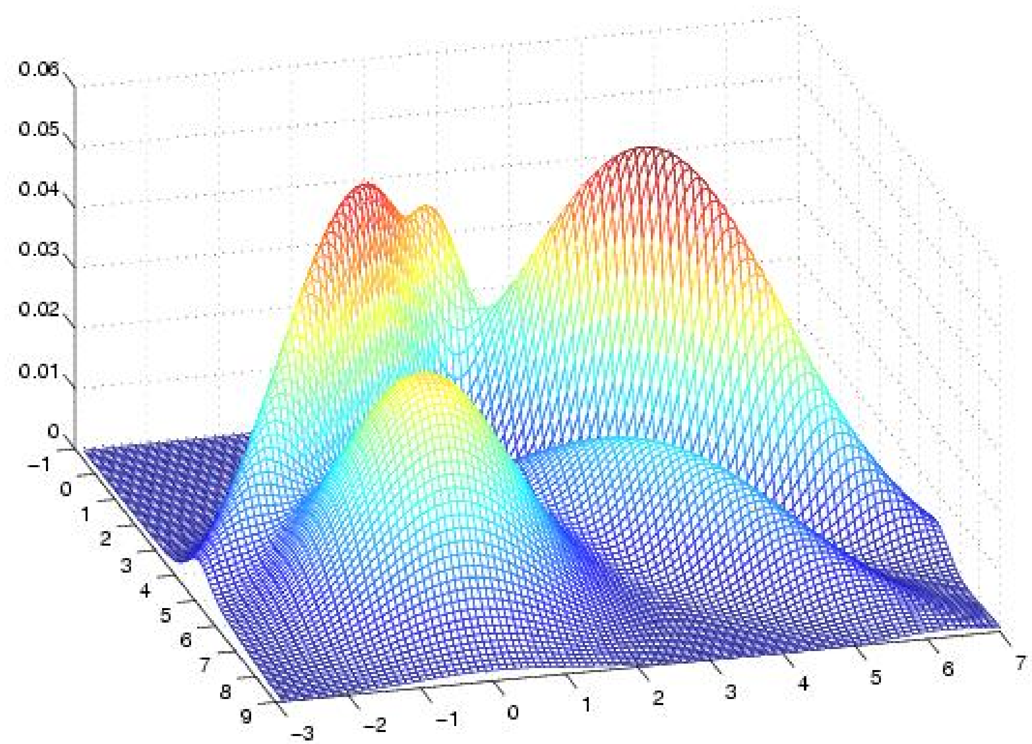
Application to clustering: Natural model if...

1. each data point belongs to exactly one group.
2. we have some idea what the group densities look like.

Gaussian mixture model

$$p\left(x|\boldsymbol{\mu},\boldsymbol{\sigma}\right)=\sum_{j=1}^{l}c_j g\left(x|\mu_j,\sigma_j\right)$$





Parametric mixtures: Inference

Inference: How can we estimate the model parameters c_j, μ_j, σ_j?

We refer to the source information (i.e.,which component was a data point drawn from) as *assignments*.

Problem:

- Parameters can be estimated by ML *if assignments are known*.
- Assignments can be estimated from model if parameters are known.

Idea: Iterative approach.

Expectation-Maximization algorithm

Estimate Gaussian mixture from data values $x_1, \ldots, x_n$.

Approach: Regard class assignments as random variables.

Notation: Assignment variables $M_{ij} := \begin{cases} 1 & x_i \text{ drawn from } p_j \\ 0 & \text{otherwise} \end{cases}$

Algorithm: Iterates two steps:

- **E-step:** Estimate expected values for M_{ij} from current model configuration.
- **M-step:** Estimate model parameters from current assignment probabilities $\mathrm{E}[M_{ij}]$.

This will require some more explanation.

Gaussian mixture: E-step

Current model parameters: $\tilde{\boldsymbol{\theta}} = (\tilde{\mathbf{c}}, \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\sigma}})$ (from last M-step)

Compute expectations:

$$\mathsf{E}\left[M_{ij}\,\middle|\mathbf{x}, \tilde{\boldsymbol{\theta}}\right] = \Pr\{x_i \text{ was drawn from } p_j\}$$

$$= \frac{c_j p(x_i|\tilde{\theta}_j)}{\sum_{k=1}^{l} c_k p(x_i|\tilde{\theta}_k)} = \frac{c_j g\left(x_i|\tilde{\mu}_j, \tilde{\sigma}_j\right)}{\sum_{k=1}^{l} c_k g\left(x_i|\tilde{\mu}_k, \tilde{\sigma}_k\right)}$$

Jargon: The binary assignments ("hard assignments") are *relaxed* to values $\mathsf{E}\left[M_{ij}\right] \in [0, 1]$ ("soft assignments").

Gaussian mixture: M-step

Task: Estimate model parameters given assignments.

Easy for hard assignments:

- Select all x_i with $M_{ij} = 1$.
- Perform ML estimation on this data subset.

Can we do it for soft assignments? The log-likelihood is

$$l_{\mathbf{M}}(\theta) = \sum_{i=1}^{n} \log \left(\sum_{j=1}^{l} M_{ij} c_j g\left(x_i | \mu_j, \sigma_j\right) \right)$$

Technical problem: We want to substitute expected values for M_{ij}. We can apply an expectation to $l_{\mathbf{M}}$, but how do we get it into the log?

Gaussian mixture: M-step

Trick: (This is a true classic.)

$$\sum_{i=1}^{n} \log\left(\sum_{j=1}^{l} M_{ij} c_j g\left(x_i|\mu_j, \sigma_j\right)\right) = \sum_{i=1}^{n}\sum_{j=1}^{l} M_{ij} \log\left(c_j g\left(x_i|\mu_j, \sigma_j\right)\right)$$

Explanation: For all i, $M_{ij_0} = 1$ for exactly one j_0. So:

$$\log\left(\sum_{j=1}^{l} M_{ij} f_j\right) = \log\left(f_{j_0}\right) = M_{ij_0} \log\left(f_{j_0}\right) = \sum_{j} M_{ij} \log\left(f_j\right)$$

Note: This introduces an error, because it is only valid for hard assignments. We relax assignments, and relaxation differs inside and outside logarithm.

Gaussian mixture: M-step

Expected log-likelihood:

$$
\begin{aligned}
\mathsf{E}_{\mathbf{M}|\mathbf{x},\tilde{\theta}}\left[l\left(\theta\right)\right] &= \mathsf{E}\left[\sum_{i=1}^{n}\sum_{j=1}^{l} M_{ij}\log\left(c_j g\left(x_i|\mu_j,\sigma_j\right)\right)\right] \\
&= \sum_{i=1}^{n}\sum_{j=1}^{l}\mathsf{E}\left[M_{ij}\right]\log\left(c_j g\left(x_i|\mu_j,\sigma_j\right)\right) \\
&= \underbrace{\sum_{i,j}\mathsf{E}\left[M_{ij}\right]\log\left(c_j\right)}_{1} + \underbrace{\sum_{i,j}\mathsf{E}\left[M_{ij}\right]\log\left(g\left(x_i|\mu_j,\sigma_j\right)\right)}_{2}
\end{aligned}
$$

- Substitute E-step results for $\mathsf{E}\left[M_{ij}\right]$.
- Maximize (1) and (2) separately w. r. t. c_j and μ_j, σ_j.

Gaussian mixture: M-step

Maximizing (1):

$$c_j := \frac{1}{n} \sum_i \mathsf{E}\left[M_{ij}\right]$$

Maximizing (2): For 1D Gaussian model, analytic maximization gives

$$\tilde{\mu}_j = \frac{\sum_{i=1}^n x_i \mathsf{E}\left[M_{ij}\right]}{\sum_{i=1}^n \mathsf{E}\left[M_{ij}\right]}$$

$$\tilde{\sigma}_j^2 = \frac{\sum_{i=1}^n (x_i - \tilde{\mu}_j)^2 \, \mathsf{E}\left[M_{ij}\right]}{\sum_{i=1}^n \mathsf{E}\left[M_{ij}\right]}$$

$\rightarrow$ weighted form of the standard ML estimators.

EM algorithm: Summary

Notation: $Q(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) := \mathsf{E}_{\mathbf{M}|\mathbf{x},\tilde{\theta}}\left[l_{\mathbf{M}}(\theta)\right]$

EM algorithm:

- **E-step:**
  1. Substitute current parameter estimates $\tilde{\boldsymbol{\theta}}$ into model.
  2. Estimate expectations $\mathsf{E}\left[M_{ij}\right]$.
  3. Substitute estimates into log-likelihood. This gives Q as function of $\boldsymbol{\theta}$.
- **M-step:**
  Parameter estimation: Maximize $Q(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}})$ w. r. t. $\boldsymbol{\theta}$.

Observation: This does not seem to be limited to a specific model (like Gaussian mixtures). Can it be generalized?

EM: General case

When can EM be applied?

If we can define hidden variables $\mathbf{M}$ such that

- The joint density $p(\mathbf{x}, \mathbf{M}|\boldsymbol{\theta})$ is known.
- Expected values of the hidden variables can be estimated from a given model configuration.
- Given estimates for the hidden variables, ML estimation is possible.

When do we want to apply EM for ML estimation? If ...

- ... ML is hard for $p(\mathbf{x}|\boldsymbol{\theta})$
- ... ML is easy for $p(\mathbf{x}, \mathbf{M}|\boldsymbol{\theta})$ when we know $\mathbf{M}$.
- ... we can efficiently compute expectations for $\mathbf{M}$.

The two log-likelihoods

The density of the augmented data $(\mathbf{x}, \mathbf{M})$ is:

$$p\left(\mathbf{x}, \mathbf{M}|\boldsymbol{\theta}\right) = p\left(\mathbf{M}|\mathbf{x}, \boldsymbol{\theta}\right) p\left(\mathbf{x}|\boldsymbol{\theta}\right)$$

This means we deal with two different log-likelihoods:

- The one we are actually interested in:

$$l\left(\boldsymbol{\theta}\right) = \log\left(p\left(\mathbf{x}|\boldsymbol{\theta}\right)\right)$$

- The one including the hidden variables:

$$l_{\mathbf{M}}\left(\boldsymbol{\theta}\right) = \log\left(p\left(\mathbf{x}, \mathbf{M}|\boldsymbol{\theta}\right)\right)$$

$l\left(\boldsymbol{\theta}\right)$ is constant w. r. t. the expectation $\mathsf{E}_{\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}}\left[\,.\,\right]$ in the algorithm. $l_{\mathbf{M}}\left(\boldsymbol{\theta}\right)$ is dependent on hidden variables $\mathbf{M}$.

Proof of Convergence

What we want to show: $l(\boldsymbol{\theta}) > l(\tilde{\boldsymbol{\theta}})$.

Rewrite $l(\boldsymbol{\theta})$ using definition of conditional prob.:

$$\begin{aligned} l(\boldsymbol{\theta}) = \log\left(p\left(\mathbf{x}|\boldsymbol{\theta}\right)\right) &= \log\left(\frac{p\left(\mathbf{x},\mathbf{M}|\boldsymbol{\theta}\right)}{p\left(\mathbf{M}|\mathbf{x},\boldsymbol{\theta}\right)}\right) \\ &= l_{\mathbf{M}}(\boldsymbol{\theta}) - \log\left(p\left(\mathbf{M}|\mathbf{x},\boldsymbol{\theta}\right)\right) \end{aligned}$$

Apply the expectation:

$$\begin{aligned} \mathsf{E}_{\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}}\left[l(\boldsymbol{\theta})\right] &= \mathsf{E}_{\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}}\left[l_{\mathbf{M}}(\boldsymbol{\theta})\right] - \mathsf{E}_{\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}}\left[\log\left(p\left(\mathbf{M}|\mathbf{x},\boldsymbol{\theta}\right)\right)\right] \\ \Leftrightarrow \qquad l(\boldsymbol{\theta}) &= Q(\boldsymbol{\theta},\tilde{\boldsymbol{\theta}}) - \mathsf{E}_{\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}}\left[\log\left(p\left(\mathbf{M}|\mathbf{x},\boldsymbol{\theta}\right)\right)\right] \end{aligned}$$

Proof of convergence

We want to show that this is larger than

$$l(\tilde{\boldsymbol{\theta}}) = Q(\tilde{\boldsymbol{\theta}}, \tilde{\boldsymbol{\theta}}) - \mathsf{E}_{\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}}\left[\log\left(p\left(\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}\right)\right)\right]$$

First term Q: Two possibilities,

1. Q is already maximal (algorithm converged).
2. Otherwise: $Q(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) > Q(\tilde{\boldsymbol{\theta}}, \tilde{\boldsymbol{\theta}})$.

For the second term holds:

$$\mathsf{E}_{\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}}\left[\log\left(p\left(\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}\right)\right)\right] \geq \mathsf{E}_{\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}}\left[\log\left(p\left(\mathbf{M}|\mathbf{x},\boldsymbol{\theta}\right)\right)\right] \quad (*)$$

Proof of convergence

Summary:

$$
\begin{aligned}
l(\boldsymbol{\theta}) &= Q(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) - \mathsf{E}_{\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}}\left[\log\left(p\left(\mathbf{M}|\mathbf{x},\boldsymbol{\theta}\right)\right)\right] \\
&> Q(\tilde{\boldsymbol{\theta}}, \tilde{\boldsymbol{\theta}}) - \mathsf{E}_{\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}}\left[\log\left(p\left(\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}\right)\right)\right] \\
&= l(\tilde{\boldsymbol{\theta}})
\end{aligned}
$$

We're done, except for $(*)$.

Proof of $(*)$**:** Use *Jensen's inequality*: If f is a convex function then $\mathsf{E}\left[f\left(X\right)\right] \geq f\left(\mathsf{E}\left[X\right]\right)$ for any RV X. The log function is concave, so $\mathsf{E}\left[\log\left(X\right)\right] \leq \log\left(\mathsf{E}\left[X\right]\right)$.

Abbreviate $p := p(\mathbf{M}|\mathbf{x}, \boldsymbol{\theta})$ and $\tilde{p} := p\left(\mathbf{M}|\mathbf{x}, \tilde{\boldsymbol{\theta}}\right)$.

$$
\begin{aligned}
\mathsf{E}_{\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}}[\log(p)] &= \mathsf{E}_{\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}}\left[\log\left(\frac{p}{\tilde{p}} \cdot \tilde{p}\right)\right] \\
&= \mathsf{E}_{\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}}\left[\log\left(\frac{p}{\tilde{p}}\right)\right] + \mathsf{E}_{\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}}[\log(\tilde{p})] \\
&\leq \log\left(\mathsf{E}_{\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}}\left[\frac{p}{\tilde{p}}\right]\right) + \mathsf{E}_{\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}}[\log(\tilde{p})] \\
&= \log\left(\sum \tilde{p} \cdot \frac{p}{\tilde{p}}\right) + \mathsf{E}_{\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}}[\log(\tilde{p})] \\
&= \log(\underbrace{\sum p}_{=1}) + \mathsf{E}_{\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}}[\log(\tilde{p})] \\
&= \mathsf{E}_{\mathbf{M}|\mathbf{x},\tilde{\boldsymbol{\theta}}}[\log(\tilde{p})] \qquad \square
\end{aligned}
$$

Convergence results

Theoretical convergence guarantees:

- What we have shown above: The log-likelihood increases with each iteration. This does not imply convergence to local maximum.
- For sufficiently regular log-likelihoods, the algorithm always converges to a *local* maximum of the log-likelihood.

What can go wrong: Like any gradient-type algorithm, it can get stuck in a saddle point or even a local minimum. Note:

- This is a *scale problem*. It happens when the gradient step is too large to resolve a local maximum and oversteps.
- Can be prevented by requiring regularity conditions.
- Only happens for numerical M-step.

Convergence in practice

Hard to analyze:

- Cost function (log-likelihood) changes between steps.
- Influence of hidden variables is not entirely understood.

Local minima/saddle points: Convergence to these points is a theoretical possibility, but usually not a practical problem.

Worst problem: Initialization. EM results tend to be highly dependent on initial values.

Common strategy: Initialize with random values. Rerun algorithm several times and choose solution which has the largest likelihood.

k-Means algorithm

Simplify Gaussian mixture model EM:

1. Assume that all Gaussians have the same variance.
2. Use hard assignments instead of expectations.

Resulting algorithm: Alternate steps

1. For each class, choose all assigned data values and average them. ($\longrightarrow$ ML estimation of Gaussian mean for hard assignments.)
2. Assign each value to class under which its probability of occurrence is largest.

Hence the name: For k classes, algorithm iteratively adjust means (= class averages).

Some history

EM: Introduced by Dempster, Laird and Rubin in 1977. Previously known as Baum-Welch algorithm for Hidden Markov Models.

k-Means: Also known as Lloyd-Max-Algorithm in vector quantization. In 1973, Bezdek introduced a ’fuzzy’ version of k-Means which comes very close to EM for mixture models.

EM convergence: Dempster, Laird and Rubin proved a theorem stating that EM always converges to a local maximum, but their proof was wrong. In 1983, Wu gave a number of regularity conditions sufficient to ensure convergence to a local maximum.