Support Vector Methods in Learning and Feature Extraction

Bernhard Schölkopf[1,2], *Alex Smola*[1,2], *Klaus-Robert Müller*[1],
Chris Burges[3], *Vladimir Vapnik*[4]

[1]GMD FIRST, Rudower Chaussee 5, 12489 Berlin, Germany
[2]Australian National University, Dept. of Engineering, Canberra
[3]Bell Laboratories, 101 Crawfords Corner Road, Holmdel NJ, USA
[4]AT&T Research, 100 Schultz Dr., Red Bank NJ, USA

bs@first.gmd.de, http://svm.first.gmd.de

Learning Pattern Recognition from Examples

Assume $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell) \in \mathbf{R}^N \times \{\pm 1\}$, drawn from $P(\mathbf{x}, y)$.

$$\text{Risk} \quad R[f] = \int \frac{1}{2}|f(\mathbf{x}) - y| \, dP(\mathbf{x}, y)$$

$$\text{Empirical Risk} \quad R_{emp}[f] = \frac{1}{\ell} \sum_{i=1}^{\ell} \frac{1}{2}|f(\mathbf{x}_i) - y_i|$$

Structural risk minimization (Vapnik, 1979):
With probability of at least η,

$$R[f] \leq R_{emp}[f] + \Phi\left(\frac{h}{\ell}, \eta\right).$$

The VC(Vapnik-Chervonenkis)-dimension h is a property of the set of functions implementable by the learning machine.
Successful learning requires controlling the VC-dimension.

Preprocessing / Feature Extraction

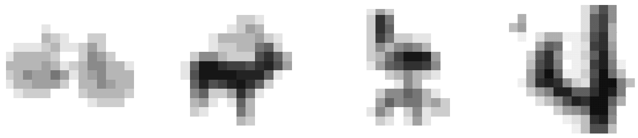
Learning f requires small training error *and* small capacity of the set $\{f_\alpha\}$.

One approach: *preprocess* the data with g, and learn f^* to construct

$$f = f^* \circ g$$

Goal: choose g such that $\{f^*_\alpha\}$ allows small training error and is "simple".

Product Feature Space



Patterns $\mathbf{x} = (x_1, \ldots, x_N) \in \mathbf{R}^N$ where most information is contained in products between entries x_i, e.g. in $x_1 \cdot x_2$.

$\longrightarrow$ work in the space F of all products of d entries (e.g. pixels).

E.g., for $N = 2$, use a map

$$\Phi : \mathbf{R}^2 \to F = \mathbf{R}^3$$
$$(x_1, x_2) \mapsto (x_1^2, x_2^2, x_1 x_2)$$

Problem: Dimensionality of F

Number of products of d pixels in N-dimensional images:

$$\frac{(N+d-1)!}{d!(N-1)!}$$

— grows like N^d.

E.g. $N = 16 \times 16$, and $d = 5 \longrightarrow$ dimensionality 10^{10}

Computing Dot Products in F by Kernels

(Aizerman, Braverman & Rozonoer, 1964, Boser, Guyon, & Vapnik, 1992)

Try to compute dot products in F with a simple kernel

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

without carrying out Φ.

Example in $\mathbf{R}^2$

$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \mapsto \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}\, x_1 x_2 \end{pmatrix}$$

Then

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

Also works for $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d$ in $\mathbf{R}^N$.

Polynomial Kernels

Define C_d to map $\mathbf{x} \in \mathbf{R}^N$ to the vector $C_d(\mathbf{x})$ of all possible d-th degree ordered products of the entries of $\mathbf{x}$. The kernel computing the dot product of vectors mapped by C_d is

$$k(\mathbf{x}, \mathbf{y}) = (C_d(\mathbf{x}) \cdot C_d(\mathbf{y})) = (\mathbf{x} \cdot \mathbf{y})^d.$$

Proof.

$$\begin{aligned} (C_d(\mathbf{x}) \cdot C_d(\mathbf{y})) &= \sum_{j_1, \ldots, j_d = 1}^{N} x_{j_1} \cdot \ldots \cdot x_{j_d} \cdot y_{j_1} \cdot \ldots \cdot y_{j_d} \\ &= \left(\sum_{j=1}^{N} x_j \cdot y_j \right)^d = (\mathbf{x} \cdot \mathbf{y})^d. \end{aligned}$$

□

Condition on k (Mercer's Theorem)

If k is a continuous kernel of a positive integral operator,

$$\int f(\mathbf{x})k(\mathbf{x},\mathbf{y})f(\mathbf{y})\,d\mathbf{x}\,d\mathbf{y} \geq 0$$

it can be expanded as

$$k(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{\infty} \lambda_i \psi_i(\mathbf{x})\psi_i(\mathbf{y})$$

with $\lambda_i \geq 0$ for $i \in \mathbf{N}$.

In that case

$$\Phi(\mathbf{x}) := \begin{pmatrix} \sqrt{\lambda_1}\psi_1(\mathbf{x}) \\ \sqrt{\lambda_2}\psi_2(\mathbf{x}) \\ \vdots \end{pmatrix}$$

satisfies $(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})) = k(\mathbf{x},\mathbf{y})$.

Another Representation of F

$$\begin{aligned} \tilde{\Phi} : \mathbf{R}^N &\longrightarrow \mathcal{H} \\ \mathbf{x} &\mapsto k(\mathbf{x}, .). \end{aligned}$$

Thus $\langle \tilde{\Phi}(\mathbf{x}), \tilde{\Phi}(\mathbf{y}) \rangle = \langle k(\mathbf{x}, .), k(\mathbf{y}, .) \rangle$. If $\langle ., . \rangle$ satisfied

$$\langle k(\mathbf{x}, .), k(\mathbf{y}, .) \rangle = k(\mathbf{x},\mathbf{y}),$$

then we would have $\langle \tilde{\Phi}(\mathbf{x}), \tilde{\Phi}(\mathbf{y}) \rangle = k(\mathbf{x},\mathbf{y})$.
For a Mercer kernel $k(\mathbf{x},\mathbf{y}) = \sum_j \lambda_j \psi_j(\mathbf{x})\psi_j(\mathbf{y})$ this can be achieved by choosing $\langle ., . \rangle$ such that

$$\langle \sqrt{\lambda_j}\psi_j, \sqrt{\lambda_n}\psi_n \rangle = \delta_{jn}.$$

$\mathcal{H}$, the closure of the space of all functions

$$f(\mathbf{x}) = \sum_i a_i k(\mathbf{x}, \mathbf{x}_i),$$

with the dot product $\langle ., . \rangle$, is called a reproducing kernel Hilbert space.

The Dot Product of the RKHS

Expand f, g in the ONB $(\sqrt{\lambda_i}\psi_i)_i$, using coordinate vectors $\boldsymbol{\alpha}, \boldsymbol{\beta}$:

$$f(\mathbf{x}) = (\boldsymbol{\alpha} \cdot \Phi(\mathbf{x})) \qquad g(\mathbf{x}) = (\boldsymbol{\beta} \cdot \Phi(\mathbf{x}))$$

Then the dot product in F equals that of the RKHS:

$$(\boldsymbol{\alpha} \cdot \boldsymbol{\beta}) = \langle f, g \rangle$$

In particular, if $\boldsymbol{\alpha}, \boldsymbol{\beta}$ are images of some points $\mathbf{s}, \mathbf{t}$ under Φ, there is a simple way to compute the dot product...

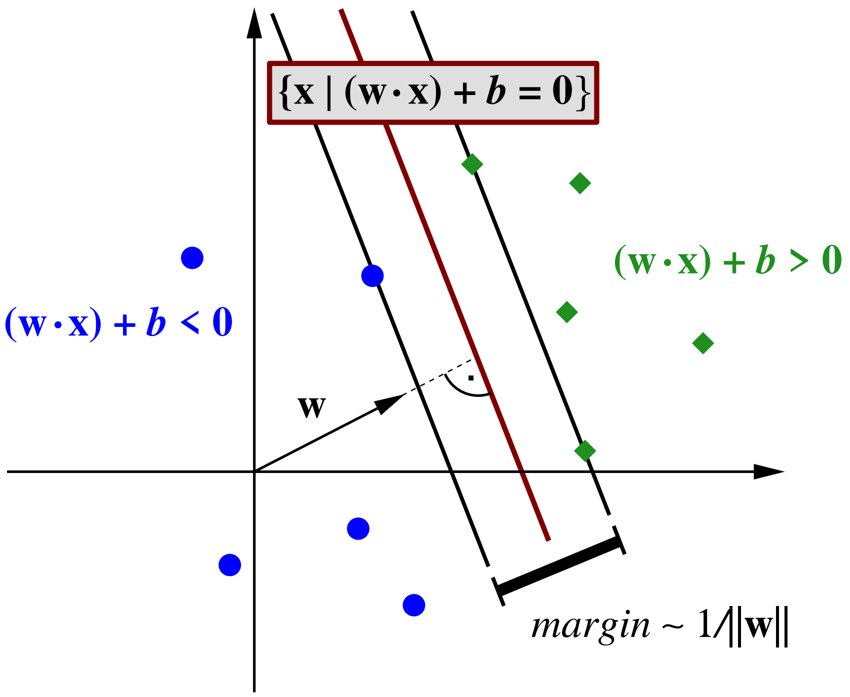
$$(\boldsymbol{\alpha} \cdot \boldsymbol{\beta}) = (\Phi(\mathbf{s}) \cdot \Phi(\mathbf{t})) = k(\mathbf{s}, \mathbf{t}).$$

Note that in this case,

$$f(\mathbf{x}) = k(\mathbf{s}, \mathbf{x}) \qquad g(\mathbf{x}) = k(\mathbf{t}, \mathbf{x})$$

Separating Hyperplanes

(Vapnik & Chervonenkis, 1974)



threshold b

weight vector $\mathbf{w}$

Maximum Margin = Minimal VC-Dimension

Formulation as an Optimization Problem

Hyperplane with maximum margin: minimize $\|\mathbf{w}\|^2$ subject to

$$y_i \cdot [(\mathbf{w} \cdot \mathbf{x}_i) + b] \geq 1 \quad \text{for } i = 1 \ldots \ell.$$

Dual: maximize

$$W(\alpha) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

subject to

$$\alpha_i \geq 0, \quad i = 1, \ldots, \ell, \quad \text{and} \quad \sum_{i=1}^{\ell} \alpha_i y_i = 0.$$

The Support Vector Expansion

Solution:

$$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i y_i \mathbf{x}_i$$

where for all $i = 1, \ldots, \ell$

$$\alpha_i = 0 \longrightarrow \mathbf{x}_i \text{ irrelevant}$$

or

$$y_i \cdot [(\mathbf{w} \cdot \mathbf{x}_i) + b] = 1 \longrightarrow \mathbf{x}_i \text{ “Support Vector” (SV)}$$

Thus

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{\ell} \alpha_i y_i (\mathbf{x} \cdot \mathbf{x}_i) + b \right).$$

Both the function to be maximized and the final decision function is expressed in dot products $\longrightarrow$ use kernels.

Nonlinear SV Machines *(Boser, Guyon & Vapnik, 1992)*

Maximize

$$W(\alpha) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$$

subject to

$$\alpha_i \geq 0, \quad i = 1, \ldots, \ell, \quad \text{and} \quad \sum_{i=1}^{\ell} \alpha_i y_i = 0.$$

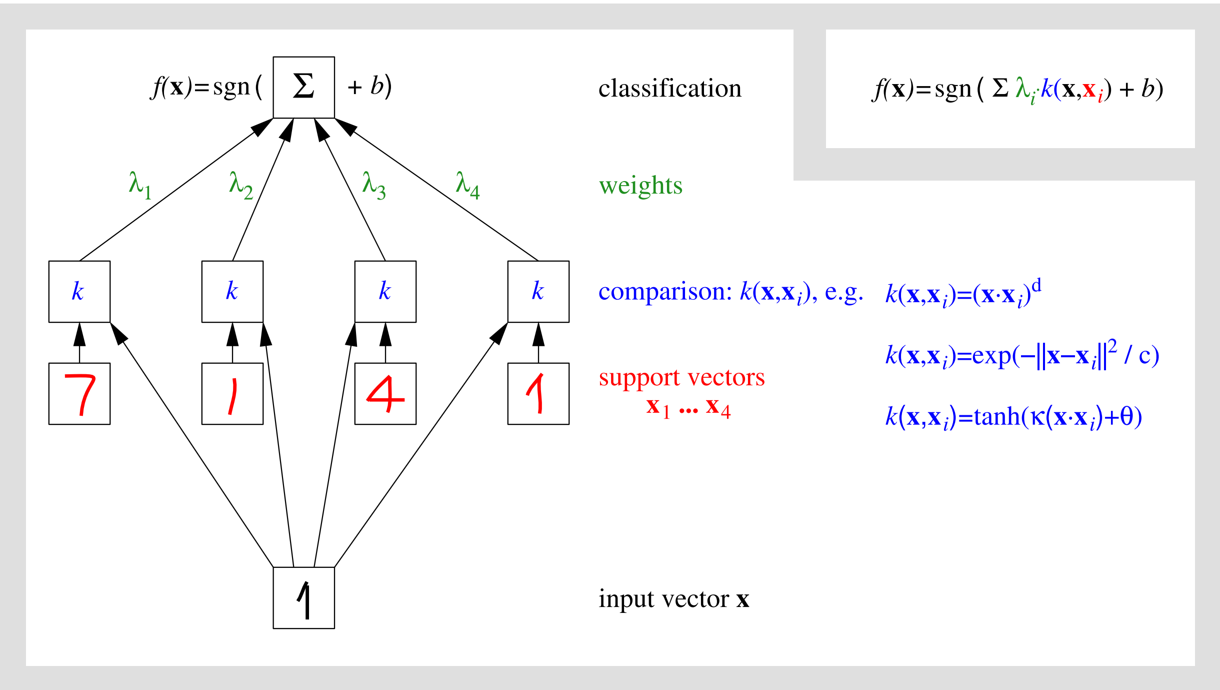
Decision function:

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{\ell} \alpha_i y_i k(\mathbf{x}, \mathbf{x}_i) + b \right)$$

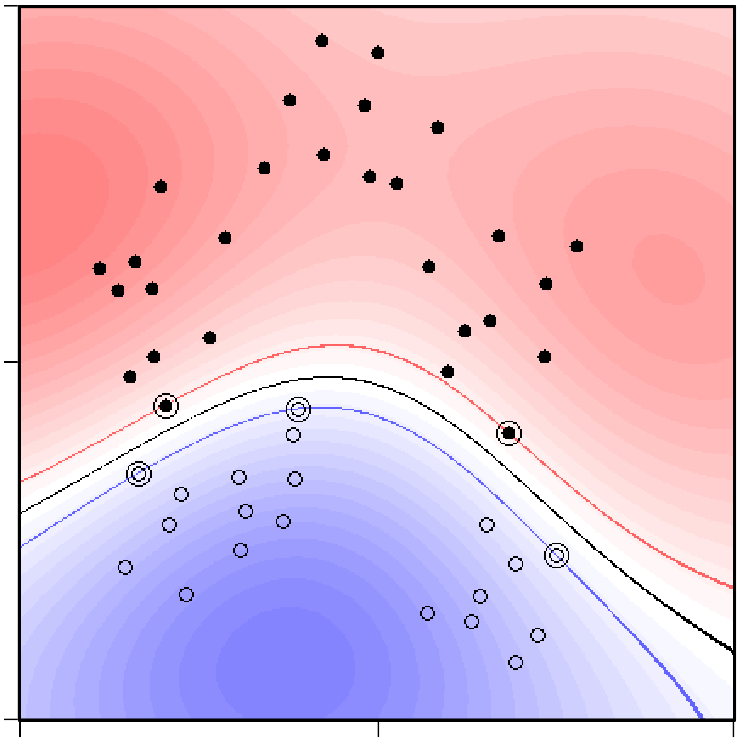
$\longrightarrow$ solve the classification problem in a high-dimensional space

Many problems become separable this way.

The Support Vector Machine



Toy Example: $k(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|^2)$



Domain: $[-1, 1]^2$

Results

(Schölkopf, Burges & Vapnik, 1995)

On a handwritten digit recognition task (US Postal Service, 7291 examples, 16×16), we constructed three types of SV classifiers:

Polynomial classifiers:	$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d$
RBF classifiers:	$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\|\mathbf{x} - \mathbf{y}\|^2/(2\sigma^2)\right)$
Neural networks:	$k(\mathbf{x}, \mathbf{y}) = \tanh(\kappa \cdot (\mathbf{x} \cdot \mathbf{y}) - \Theta)$

Empirical Results:

- they use almost the same ($\approx$ 90%) small sets ($\approx$ 4% of the data base) of Support Vectors. In the RBF case, these are the *centers*, chosen in a task-dependent way.
- all three systems perform very well ($\approx$ 4% error, LeNet1: 5%)

Using methods for incorporating prior knowledge, we obtained 3%.

SV Machines: Crucial Aspects

Computational The training of an SV machine comprises a (possibly large) convex quadratic program with simple constraints.

Statistical The kernel method allows to reduce a large class of learning machines to separating hyperplanes in some reproducing kernel space. For these, capacity can be controlled independent of dimensionality.

SVM vs. Regularization Theory *(Smola & Schölkopf, 1997)*

The problem of minimizing the regularized risk

$$R_{reg}[f] = R_{emp}[f] + \lambda \|Pf\|^2,$$

for

$$f(\mathbf{x}) = \sum_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b$$

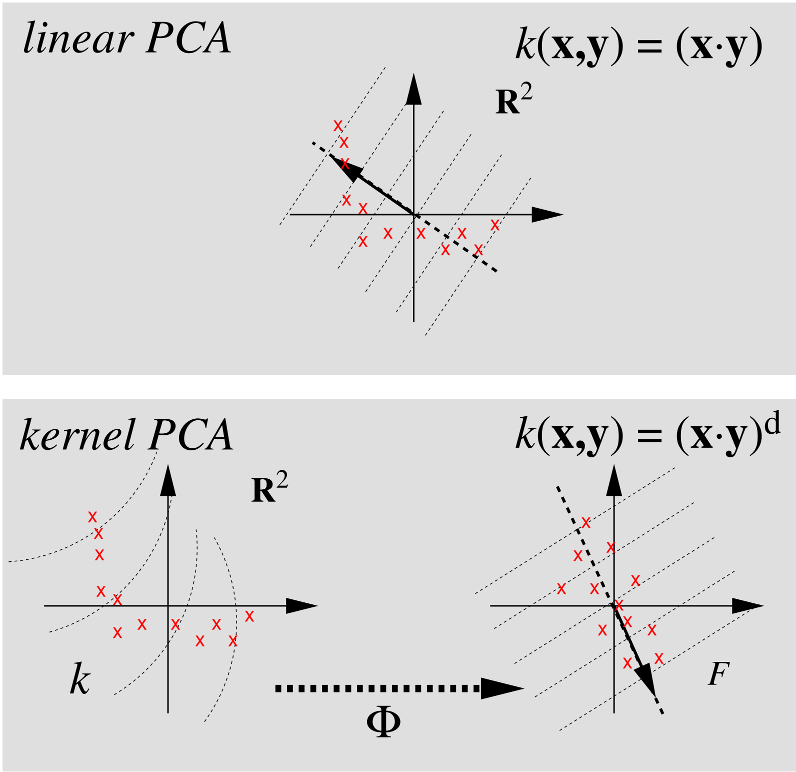
leads to a QP with matrix $KD^{-1}K$, where $D_{ij} := ((Pk)(\mathbf{x}_i, .) \cdot (Pk)(\mathbf{x}_j, .))$ and $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$.

This is equivalent to the SVM if $K = D$, which holds if k is Green's function of P^*P:

$$\begin{aligned} ((Pk)(\mathbf{x}_i, .) \cdot (Pk)(\mathbf{x}_j, .)) &= (k(\mathbf{x}_i, .) \cdot (P^*Pk)(\mathbf{x}_j, .)) \\ &= (k(\mathbf{x}_i, .) \cdot \delta_{\mathbf{x}_j}(.)) = k(\mathbf{x}_i, \mathbf{x}_j) \end{aligned}$$

Kernel PCA

(Schölkopf, Smola & Müller, 1996)



PCA in High-dimensional Feature Spaces

$\mathbf{x}_1, \ldots, \mathbf{x}_\ell, \quad \Phi : \mathbf{R}^N \to F, \ \mathbf{x} \mapsto \mathbf{X}, \quad C = \frac{1}{\ell} \sum_{j=1}^{\ell} \Phi(\mathbf{x}_j)\Phi(\mathbf{x}_j)^\top$

Eigenvalue problem

$$\lambda \mathbf{V} = C\mathbf{V} = \frac{1}{\ell} \sum_{j=1}^{\ell} (\Phi(\mathbf{x}_j) \cdot \mathbf{V})\Phi(\mathbf{x}_j).$$

For $\lambda \neq 0$, $\mathbf{V} \in \mathrm{span}\{\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_\ell)\}$, thus

$$\Longleftrightarrow \lambda(\Phi(\mathbf{x}_k) \cdot \mathbf{V}) = (\Phi(\mathbf{x}_k) \cdot C\mathbf{V}) \text{ for all } k = 1, \ldots, \ell$$

$$\mathbf{V} = \sum_{i=1}^{\ell} \alpha_i \Phi(\mathbf{x}_i)$$

Nonlinear PCA as an Eigenvalue Problem

Define an $\ell \times \ell$ matrix

$$K_{ij} := (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)) = k(\mathbf{x}_i, \mathbf{x}_j)$$

to get

$$\ell \lambda K \boldsymbol{\alpha} = K^2 \boldsymbol{\alpha}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_\ell)^\top$.
Solve

$$\ell \lambda \boldsymbol{\alpha} = K \boldsymbol{\alpha}$$

$\longrightarrow (\lambda_k, \boldsymbol{\alpha}^k)$

$$(\mathbf{V}^k \cdot \mathbf{V}^k) = 1 \Longleftrightarrow \lambda_k (\boldsymbol{\alpha}^k \cdot \boldsymbol{\alpha}^k) = 1$$

Centering in F

Center the data in F:

$$\tilde{\Phi}(\mathbf{x}_i) := \Phi(\mathbf{x}_i) - \frac{1}{\ell} \sum_{i=1}^{\ell} \Phi(\mathbf{x}_i)$$

For $\tilde{\Phi}(\mathbf{x}_i)$, everything works fine.

Express $\tilde{K}$ in terms of K, using $(1_\ell)_{ij} := 1/\ell$:

$$\tilde{K}_{ij} = K - 1_\ell K - K 1_\ell + 1_\ell K 1_\ell.$$

Compute $\tilde{K}$ and solve the Eigenvalue problem.

Similar for feature extraction.

Feature extraction

Compute projections on the Eigenvectors

$$\mathbf{V}^k = \sum_{i=1}^{M} \alpha_i^k \Phi(\mathbf{x}_i)$$

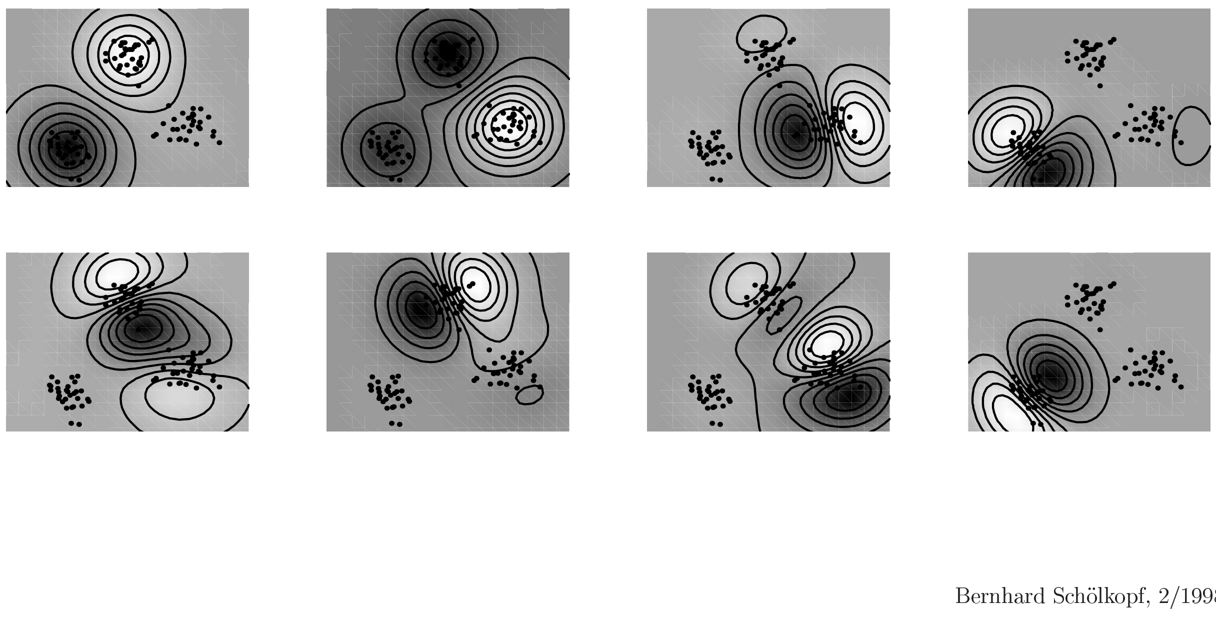
in F:

for a test point $\mathbf{x}$ with image $\Phi(\mathbf{x})$ in F we get the features

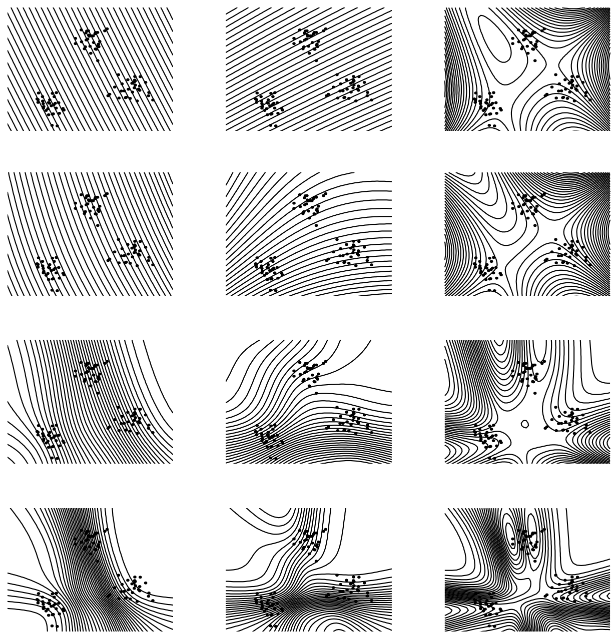
$$\begin{aligned}(\mathbf{V}^k \cdot \Phi(\mathbf{x})) &= \sum_{i=1}^{M} \alpha_i^k (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x})) \\ &= \sum_{i=1}^{M} \alpha_i^k k(\mathbf{x}_i, \mathbf{x})\end{aligned}$$

Toy Example: RBF Kernel, 8 Principal Components

$$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{0.1}\right)$$



Gradual Transition from Linear PCA to Kernel PCA



$$k(\mathbf{x}, \mathbf{y}) = \tanh\left(\kappa(\mathbf{x} \cdot \mathbf{y}) + 1\right)$$

Handwritten Digit Feature Extraction

USPS data, polynomial kernel $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d$ with $d = 1, \ldots, 7$. Linear SV machine (hyperplane) trained on kernel PCA features.

# of components	Test Error Rate for degree						
	1	2	3	4	5	6	7
64	8.8	7.3	6.8	6.7	6.7	7.2	7.5
128	8.6	5.8	5.9	6.1	5.8	6.0	6.8
256	8.7	5.5	5.3	5.2	5.2	5.4	5.4
512	n.a.	4.9	4.6	4.4	5.1	4.6	4.9
1024	n.a.	4.9	4.3	4.4	4.6	4.8	4.6
2048	n.a.	4.9	4.2	4.1	4.0	4.3	4.4

Bottomline: better than linear PCA for corresponding numbers of features, even better when freedom to use *more* features is used.

Conclusion

- *Kernels:*
  - carry out linear methods in high-dimensional spaces with a computational cost independent of their dimensionality
  - every algorithm that can be formulated in dot products could benefit from this technique
- *SV machines:*
  - extract SVs characterizing a given task
  - achieve high classification accuracies
  - prior knowledge further improves accuracies
- *Kernel PCA:*
  - extracts features leading to high classification accuracies
  - is carried out by solving an Eigenvalue problem