Computational Intelligence: Methods and Applications

Lecture 25
Kernel methods

Włodzisław Duch
SCE, NTU, Singapore
Google: Duch

Kernels!

Kernel trick: if vectors are transformed using some function (usually non-linear) into high-dimensional space separation of data may be easier to achieve. Replace:

$$\mathbf{X} \to \Phi(\mathbf{X})$$

This leads to the same problem formulation, except that X is replaced everywhere by Φ(X); in particular Lagrangian contains scalar products:

$$\mathbf{X}^{(i)} \cdot \mathbf{X}^{(j)} \to \Phi\left(\mathbf{X}^{(i)}\right) \cdot \Phi\left(\mathbf{X}^{(j)}\right) = K\left(\mathbf{X}^{(i)}, \mathbf{X}^{(j)}\right)$$

These scalar products are calculated between vectors in some transformed space; instead of calculating them directly it is sufficient to define a kernel function K(X,Y).

What kind of functions correspond to scalar products in Hilbert spaces?

They should be symmetric; formal conditions have been found in mathematical analysis by Mercer; they may influence convergence.

Kernel example

Simplest: polynomial kernel: $K(\mathbf{X}, \mathbf{Y}) = (1 + \mathbf{X} \cdot \mathbf{Y})^d$

Example: quadratic kernel in 2-D

$$K(\mathbf{X}, \mathbf{Y}) = (1 + X_1 Y_1 + X_2 Y_2)^2$$
$$= 1 + 2X_1 Y_1 + 2X_2 Y_2 + (X_1 Y_1)^2 + (X_2 Y_2)^2 + 2X_1 Y_1 X_2 Y_2$$

Use of this kernel is equivalent to working in 5-D space:

$$\mathbf{X} = (X_1, X_2) \Rightarrow \Phi(\mathbf{X}) = \left(1, \sqrt{2}X_1, \sqrt{2}X_2, X_1^2, X_2^2, \sqrt{2}X_1 X_2\right)$$

Hyperplane in 5D found using linear SVM corresponds to quadratic function in 2D; try to show that quadratic border in (X_1, X_2) space becomes a hyperplane in kernel space.

Selection of kernel may strongly influence results.

Other popular kernels

Some popular kernels working as scalar products:

$$\mathbf{X}^{(i)} \cdot \mathbf{X}^{(j)} \to \Phi\left(\mathbf{X}^{(i)}\right) \cdot \Phi\left(\mathbf{X}^{(j)}\right) = K\left(\mathbf{X}^{(i)}, \mathbf{X}^{(j)}\right)$$

Gaussian: $K_G(\mathbf{X}, \mathbf{Y}) = \exp\left(-\|\mathbf{X} - \mathbf{Y}\|^2 / 2\sigma^2\right)$

Sigmoidal: $K_s(\mathbf{X}, \mathbf{Y}) = \tanh\left(\kappa_1 \mathbf{X} \cdot \mathbf{Y} + \kappa_2\right)$

Distance: $K_d(\mathbf{X}, \mathbf{Y}) = \|\mathbf{X} - \mathbf{Y}\|^b$

Dimensionality of the **Φ** space: number of independent polynomial products or number of training vectors.

Distance kernel: for *b*=2 Euclidean distance ⇔ linear case!

In complex cases (ex. protein comparison) kernel = similarity function, especially designed for the problem.

Examples

SMO, Sequential Multiple Optimization algorithm for SVM with polynomial kernels, is implemented in WEKA/Yale and GM 1.5.

The only user adjustable parameters for polynomial version are the margin-related parameter *C* and the degree of the polynomial kernel. In GM optimal value of C may automatically be found by crossvalidation training (but it may be costly).

Note that data should be standardized to avoid convergence problems.

For other kernel functions – Gaussians, sigmoidal and exponential – additional parameters of kernels may be adjusted (GM) by the user.

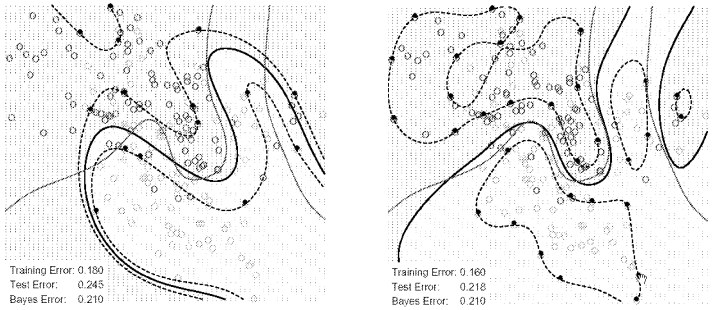
Example 1: Gaussians data clusters with some kernels

Example 2: Cleveland Heart data

Example 3: Ljubliana cancer data

Example 1: Gaussian mixtures

Gaussian kernels work quite well, giving close to optimal Bayesian error (that may be computed only because we know the distributions, but it is not exact, since finite number of points is given).

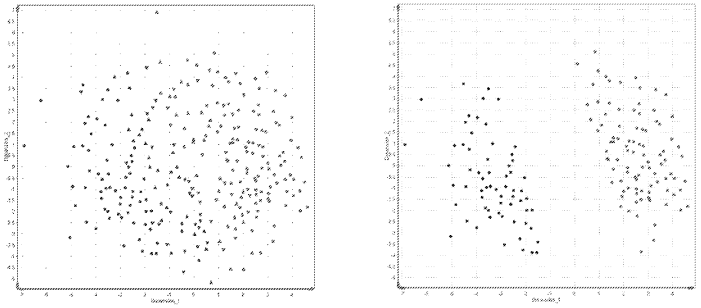


4-deg. polynomial kernel is very similar to a Gaussian kernel, C=1.

Example 2: Cleveland heart data

Left: 2D MDS features, linear SVM, C=1, acc. 81.9%

Right: support vectors removed, margin is clear, all vector inside are SV.



Gaussian kernel, C=10000, 10xCV, 100% train, 79.3 ± 7.8% test
Gaussian kernel, C=1, 10xCV, 93.8% train, 82.6 ± 8.0% test
Auto C=32 and Gaussian dispersion 0.004: about 84.4 ± 5.1% on test

Example 3: Ljubliana cancer recurrence

286 events: 85 recurrence (70.3%) and 201 no recurrence (29.7%);
9 features: tumor-size, inv-nodes, deg-malig, etc ...

Linear kernel, C=1 (C=10 similar, C=100 hard to converge):
whole data 75 errors, or 73.8%
10xCV: training 73.7±1.0%, test 71.1±8.3%

Linear kernel, C=0.01:
10xCV: training 70.6±0.7%, test 70.3±1.4% (base rate !)

Polynomial kernel k=3, C=10 (opt):
10xCV: training 89.8±0.6%, test 74.2±7.9% (best for polynomial kernel)

Gaussian kernel, opt C=1 and σ=1/4
10xCV: training 88.0±3.4%, test 74.8±6.5% (best for polynomial kernel)

But a rule: Involved Nodes > 0 & Degree_malig = 3 has 77.1% accuracy!

Some applications

SVM found many applications, see the list at:
http://www.clopinet.com/isabelle/Projects/SVM/applist.html

A few interesting applications, with highly competitive results:

- On-line Handwriting Recognition, zip codes
- 3D object recognition
- Stock forecasting
- Intrusion Detection Systems (IDSs)
- Image classification
- Detecting Steganography in digital images
- Medical applications: diagnostics, survival rates ...
- Technical: Combustion Engine Knock Detection
- Elementary Particle Identification in High Energy Physics
- Bioinformatics: protein properties, genomics, microarrays
- Information retrieval, text categorization

Get kernelized!

Discriminant function – just replace dot product by kernel:

$$g_{\mathbf{W}}(\mathbf{X}) = \sum_{i=1}^{n_{sv}} \alpha_i K\left(\mathbf{X}^{(i)}, \mathbf{X}\right) + W_0$$

now α may be negative to avoid mentioning Y_i

Number of support vectors in a separable case is small, but in non-separable case may get large – all between the margins + errors.

Kernels may be used in many discriminant methods, for example Kernel PCA or Kernel Fisher Discriminant Analysis.

Covariance matrix after transformation:

$$\mathbf{X} \to \mathbf{\Phi}(\mathbf{X}); \quad \sum_{i=1}^{n} \mathbf{\Phi}(\mathbf{X}) = \mathbf{0}$$

$\Phi(\mathbf{X})$ is d-dim vector, and

$$\mathbf{C} = \frac{1}{n-1} \sum_{i=1}^{n} \mathbf{\Phi}\left(\mathbf{X}^{(i)}\right) \mathbf{\Phi}\left(\mathbf{X}^{(i)}\right)^{\mathrm{T}} = \frac{1}{n-1} \mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}$$

$\mathbf{\Phi}$ is d x n matrix

Kernel PCA

Eigenvalues and eigenvectors of the covariance matrix: $\mathbf{CZ} = \mathbf{Z\Lambda}$

Z eigenvectors are a combination of the training vectors in $\mathbf{\Phi}$ space, so:

$$\mathbf{Z} = \mathbf{\Phi\alpha}$$

$\mathbf{\Phi}$ is d x n, $\boldsymbol{\alpha}$ coefficients are n x d

$$\mathbf{CZ} = \frac{1}{n-1} \mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi\alpha} = \mathbf{\Phi\alpha\Lambda}$$

$$\mathbf{K} = \frac{1}{n-1} \mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}$$

$\mathbf{K}$ is n x n matrix of scalar products or kernel values $\mathrm{K}_{ij}=\mathrm{K}(\mathrm{X}^{(i)},\mathrm{X}^{(j)})$

$$\mathbf{K\alpha} = \boldsymbol{\alpha\Lambda}$$

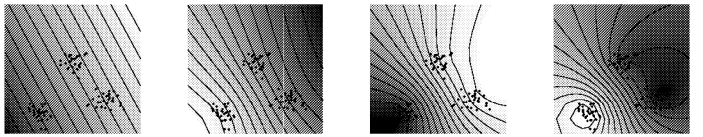
Kernel PCA coefficients are obtained from the eigenequation, and Z vectors are obtained by linear combinations; d is dim. in the Φ-space!

We have non-linear PCA using linear methods!
Good for classification, dimensionality reduction and visualization.

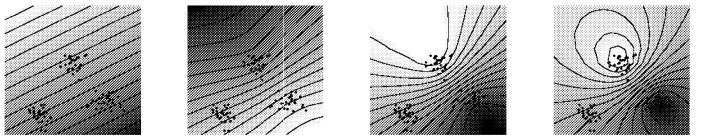
From linear to non-linear PCA

$||X-Y||^b$ kernel was used to illustrate how the constant values of the first 2 components (features) change from straight lines for the linear PCA (b=2), to a non-linear ones for b=1.5, 1 and 0.5.

1st PCA



2nd PCA



From Schölkopf and Smola, Learning with kernels, MIT 2002

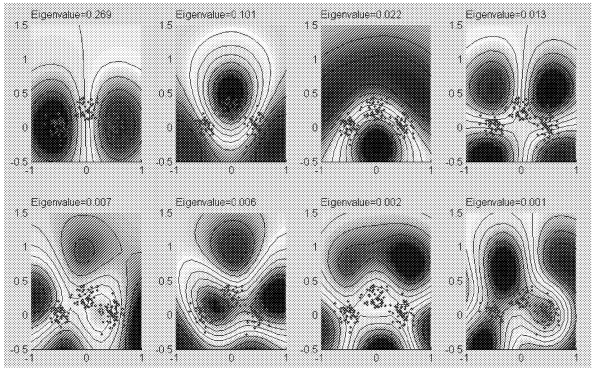
$$PC_k(\mathbf{X}) = \sum_{i=1}^{N_{SV}} \alpha_{ki} K\left(\mathbf{X}^{(i)}, \mathbf{X}\right)$$

Kernel PCA for Gaussians

2-D data with 3 Gaussians; kernel PCA was performed with Gaussian kernels. Constant value of features follow the cluster densities!

First two kernel PCs separate the data nicely.

(made with B. Schölkopf Matlab program)



Linear PCA has only 2 components, but kernel PCA has more, since the **Φ** space dimension is usually large - here each Gaussian with $||X^{(i)}-X||$, i=1..n is one function. Example: use this Matlab program.

Kernel FDA

Fisher criterion for linear discrimination was used for classification and visualization; for two classes:

$$\max_{\mathbf{W}} J(\mathbf{W}) = \frac{\mathbf{W}^{\mathrm{T}}\mathbf{S}_B\mathbf{W}}{\mathbf{W}^{\mathrm{T}}\mathbf{S}_I\mathbf{W}} \qquad \mathbf{S}_B = \left(\bar{\mathbf{X}}_1 - \bar{\mathbf{X}}_2\right)\left(\bar{\mathbf{X}}_1 - \bar{\mathbf{X}}_2\right)^{\mathrm{T}}$$

$$\mathbf{S}_I = \sum_{k=1}^{2}\sum_{j=1}^{n(C_k)}\left(\mathbf{X}^{(j)} - \bar{\mathbf{X}}_k\right)\left(\mathbf{X}^{(j)} - \bar{\mathbf{X}}_k\right)^{\mathrm{T}}$$

Projecting X on the **Φ** space: $\mathbf{W} = \sum_{i=1}^{n}\alpha_i\Phi\left(\mathbf{X}^{(i)}\right) = \mathbf{\Phi}^{\mathrm{T}}\cdot\boldsymbol{\alpha}$

Matrix of scalar products or kernel values $K_{ij}=K(X^{(i)},X^{(j)})$

Scatter matrices are calculated now in the Φ space, Kernel Fisher criterion is: $\max_{\boldsymbol{\alpha}} J(\boldsymbol{\alpha}) = \frac{\boldsymbol{\alpha}^{\mathrm{T}}\mathbf{\Phi}_B\boldsymbol{\alpha}}{\boldsymbol{\alpha}^{\mathrm{T}}\mathbf{\Phi}_I\boldsymbol{\alpha}}$

but matrices are much bigger now; faster numerical solution is found via quadratic optimization formulation.

Further reading

Kernel methods were already introduced long time ago:

M. Aizerman, E. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. Automation and Remote Control, 25, 821-837, 1964.

Also quadratic optimization has been used in Adatron algorithm developed in statistical mechanics already in 1964; large-margin classifiers, slack variables and other ideas are also from 1960 ...

Modern revival of these ideas:

V.N. Vapnik, Statistical Learning Theory. Wiley, New York, 1998.

B. Schölkopf, A.J. Smola, Learning with kernels. MIT Press 2001.

Best source of tutorials, software and links:

http://www.kernel-machines.org/