Advanced Computational Econometrics: Machine Learning

Chapter 8: Kernel-based Regression and Feature Extraction

Thierry Denœux

July-August 2019

Kernel methods

- In the previous chapter, we have seen that the "kernel trick" makes it possible to define new features implicitly using a kernel function, which computes a dot product in a transformed feature space $\mathcal{H}$.
- A SVM then performs linear classification in $\mathcal{H}$.
- The same "kernel trick" can be applied to any learning method (supervised or unsupervised), which uses only dot products between training vectors.
- In this chapter, we will see two applications of this principle:
  1. Support-Vector regression (SVR)
  2. Kernel Principal Component Analysis (KPCA)

Overview

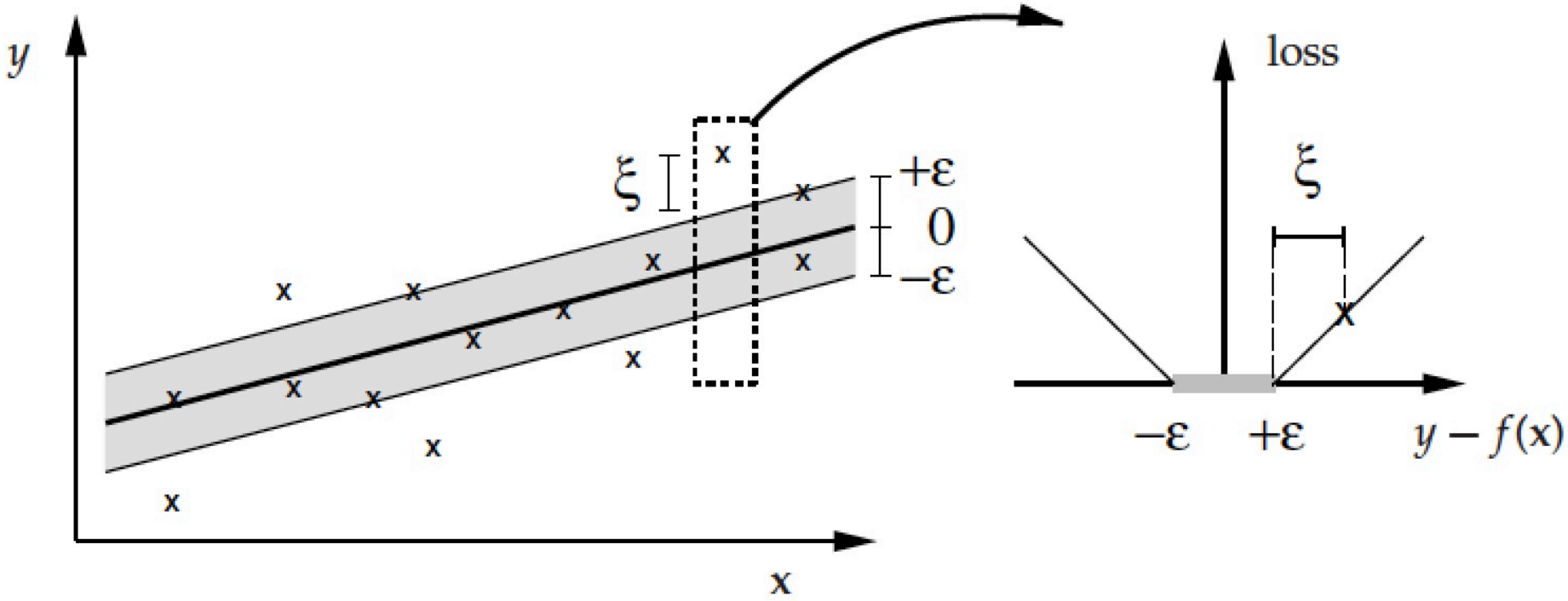
Overview

From classification to regression

- SVMs were first developed for classification.
- As described in the previous chapter, they represent the decision boundary in terms of a typically small subset of all training examples – the Support Vectors.
- To generalize the SV algorithm to regression, we need to find a way of retaining this feature. This can be achieved using the ϵ-insensitive loss function

$$
\begin{aligned}
|f(x) - y|_\epsilon &= \max(0, |f(x) - y| - \epsilon) \\
&= \begin{cases} 0 & \text{if } |f(x) - y| \leq \epsilon, \\ |f(x) - y| - \epsilon & \text{otherwise.} \end{cases}
\end{aligned}
$$

ϵ-insensitive loss function



The ϵ-insensitive loss function does not penalize errors below some ϵ, chosen a priori. The ϵ-insensitive zone is sometimes referred to as the ϵ-tube.

Basic approach

- The regression algorithm is then developed in close analogy to the case of classification.
- Again, we estimate linear functions, use a $\|\beta\|^2$ regularizer, and rewrite everything in terms of dot products to generalize to the nonlinear case.

Overview

Problem formulation

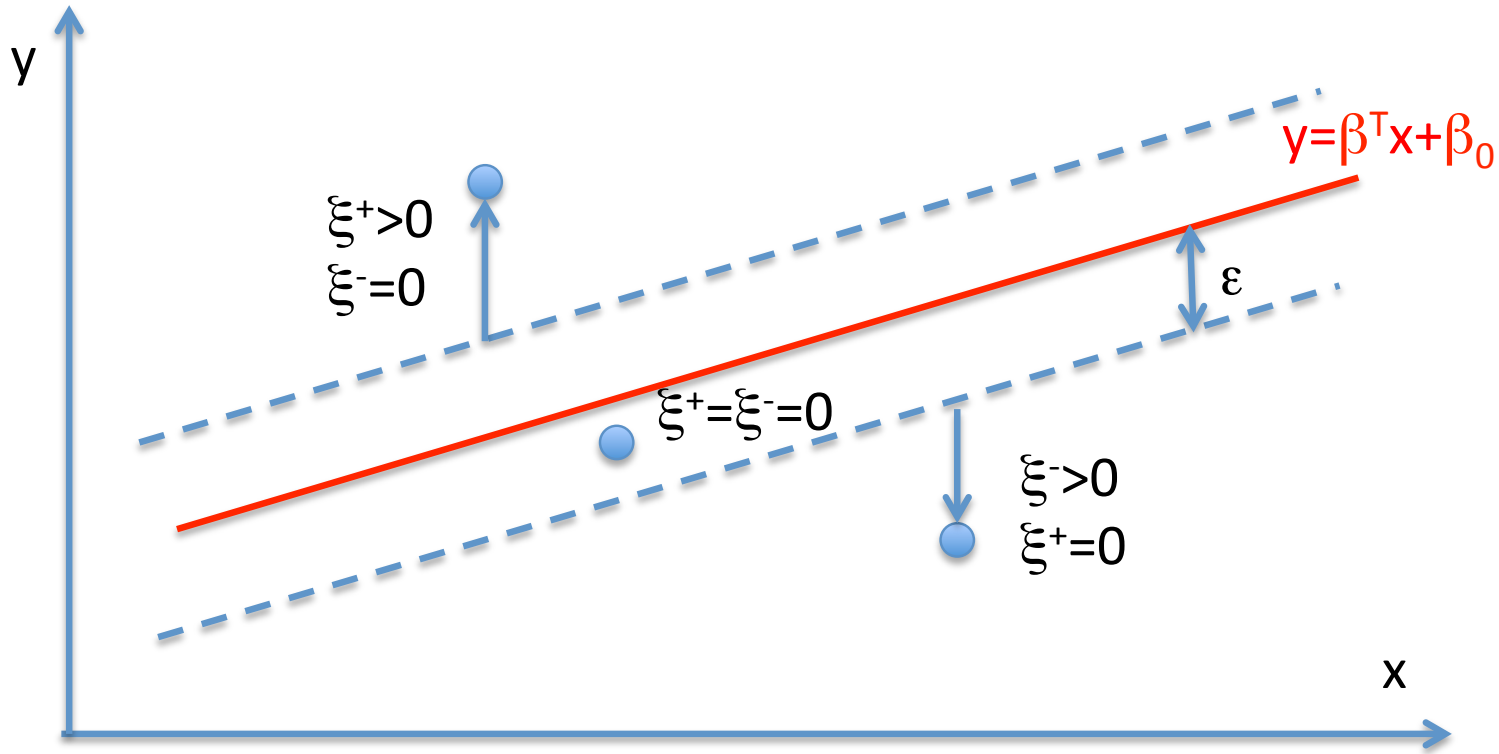
- We search for the linear function $f(x) = \beta^T x + \beta_0$ minimizing the following criterion:

$$\frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n} |f(x_i) - y_i|_\epsilon.$$

- The first term is a regularization term, which penalizes more complex functions f.
- The second term is the empirical loss (training error).
- C is a hyperparameter, which balances training error and model complexity.
- To solve this problem, we transform it into an equivalent constrained optimization problem.

Reformulation as a constrained optimization problem

- We want most observations y_i to be in the tube.
- We introduce slack variables $\xi_i^+ = [y_i - (\beta^T x_i + \beta_0 + \epsilon)]_+$ and $\xi_i^- = [\beta^T x_i + \beta_0 - \epsilon - y_i]_+$, $i = 1, \ldots, n$.



Primal objective function

Using the slack variables, the previous problem can be reformulated as a quadratic optimization problem:

$$\min_{\beta,\beta_0,\xi^-,\xi^+} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n}(\xi_i^- + \xi_i^+)$$

subject to:

$$\begin{aligned}
\xi_i^+ &\geq y_i - \beta^T x_i - \beta_0 - \epsilon \\
\xi_i^+ &\geq 0 \\
\xi_i^- &\geq \beta^T x_i + \beta_0 - y_i - \epsilon \\
\xi_i^- &\geq 0
\end{aligned}$$

for $i = 1, \ldots, n$.

Lagrange function

The Lagrange function is

$$L_P(\beta, \beta_0, \alpha_i^-, \alpha_i^+, \eta_i^-, \eta_i^+) = \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n}(\xi_i^- + \xi_i^+)$$
$$- \sum_{i=1}^{n}(\eta_i^-\xi_i^- + \eta_i^+\xi_i^+) - \sum_{i=1}^{n}\alpha_i^-(\epsilon + \xi_i^- + y_i - \beta^T x_i - \beta_0)$$
$$- \sum_{i=1}^{n}\alpha_i^+(\epsilon + \xi_i^+ - y_i + \beta^T x_i + \beta_0),$$

where α_i^-, α_i^+, η_i^-, η_i^+ are Lagrange multipliers.

Derivatives of the Lagrange function

Setting the derivatives to zero, we obtain:

$$\begin{aligned}
\frac{\partial L_P}{\partial \beta} &= \beta - \sum_{i=1}^{n}(\alpha_i^+ - \alpha_i^-)x_i = 0, \\
\frac{\partial L_P}{\partial \beta_0} &= \sum_{i=1}^{n}(\alpha_i^- - \alpha_i^+) = 0, \\
\frac{\partial L_P}{\partial \xi_i^-} &= C - \alpha_i^- - \eta_i^- = 0, \quad i = 1, \ldots, n \\
\frac{\partial L_P}{\partial \xi_i^+} &= C - \alpha_i^+ - \eta_i^+ = 0, \quad i = 1, \ldots, n.
\end{aligned}$$

Dual problem

Using these relations, we get the following Wolfe dual function:

$$L_D(\alpha_i^-, \alpha_i^+) = -\frac{1}{2}\sum_{i,j}(\alpha_i^+ - \alpha_i^-)(\alpha_j^+ - \alpha_j^-)x_i^T x_j - \epsilon\sum_{i=1}^{n}(\alpha_i^+ + \alpha_i^-) + \sum_{i=1}^{n} y_i(\alpha_i^+ - \alpha_i^-),$$

to be maximized subject to

$$\sum_{i=1}^{n}(\alpha_i^+ - \alpha_i^-) = 0$$

$$0 \leq \alpha_i^- \leq C, \quad i = 1, \ldots, n,$$

$$0 \leq \alpha_i^+ \leq C, \quad i = 1, \ldots, n.$$

Overview

Support vectors

- As in the case of SVMs, the dual problem can be solved using any quadratic programming solver.
- Let $\alpha_i^{-*}, \alpha_i^{+*}$, $i = 1, \ldots, n$ be the solution.
- The learning vectors x_i such that $\alpha_i^{-*} > 0$ or $\alpha_i^{+*} > 0$ are called the support vectors. They lie outside the tube (or at the border).
- Let S be the set of support vectors. We have

$$\beta^* = \sum_{i \in S} (\alpha_i^{+*} - \alpha_i^{-*}) x_i$$

and

$$f^*(x) = \sum_{i \in S} (\alpha_i^{+*} - \alpha_i^{-*}) x_i^T x + \beta_0^*$$

Sparsity of the SV expansion

- We have a sparse expansion of β in terms of x_i (we do not need all x_i to describe β).
- The points inside the tube (i.e., which are not support vectors) do not contribute to the solution: we could remove any one of them, and still obtain the same solution.

Karush-Kuhn-Tucker conditions

- The solution $\alpha_i^{-*}, \alpha_i^{+*}$, $i = 1, \ldots, n$ must satisfy the KKT conditions

$$\alpha_i^{-*}(\epsilon + \xi_i^{-*} + y_i - \beta^{*T} x_i - \beta_0^*) = 0 \tag{1a}$$

$$\alpha_i^{+*}(\epsilon + \xi_i^{+*} - y_i + \beta^{*T} x_i + \beta_0^*) = 0 \tag{1b}$$

$$(C - \alpha_i^{-*})\xi_i^{-*} = 0, \quad (C - \alpha_i^{+*})\xi_i^{+*} = 0 \tag{1c}$$

- Consequences:
  - Only examples (x_i, y_i) with corresponding $\alpha_i^{-*} = C$ or $\alpha_i^{+*} = C$ can lie outside the tube (i.e., $\xi_i^{-*} > 0$ or $\xi_i^{+*} > 0$).
  - For $\alpha_i^{+/-*} \in (0, C)$ we have $\xi_i^{+/-*} = 0$. The corresponding SVs lie at the border of the tube.

Calculation of β_0

- β_0^* can be calculated from (1a) or (1b) as follows

$$\beta_0^* = \begin{cases} \epsilon + y_i - \beta^{*T} x_i & \text{for } \alpha_i^{-*} \in (0, C) \\ y_i - \beta^{*T} x_i - \epsilon & \text{for } \alpha_i^{+*} \in (0, C) \end{cases}$$

- Theoretically, it suffices to use any Lagrange multiplier in $(0, C)$.
- If given the choice between several such multipliers (usually there are many multipliers which are not 'at bound', meaning that they do not equal 0 or C), it is safest to use one that is not too close to 0 or C.

Parameter tuning

The solution depends on two parameters, ϵ and C. These play different roles:

- Parameter ϵ in the loss function specifies the desired accuracy of the approximation. If we scale our response, then we might consider using preset values for ϵ.
- The quantity C is a more traditional regularization parameter. It can be estimated, for example, by cross-validation.

Nonlinear extension

- As in the classification case, the complete algorithm can be described in terms of dot products between the data.
- This makes it possible to formulate a nonlinear extension using kernels, replacing dot products $x_i^T x_j$ in $\mathcal{X}$ with dot products

$$\langle \Phi(x_i), \Phi(x_j) \rangle = \mathcal{K}(x_i, x_j)$$

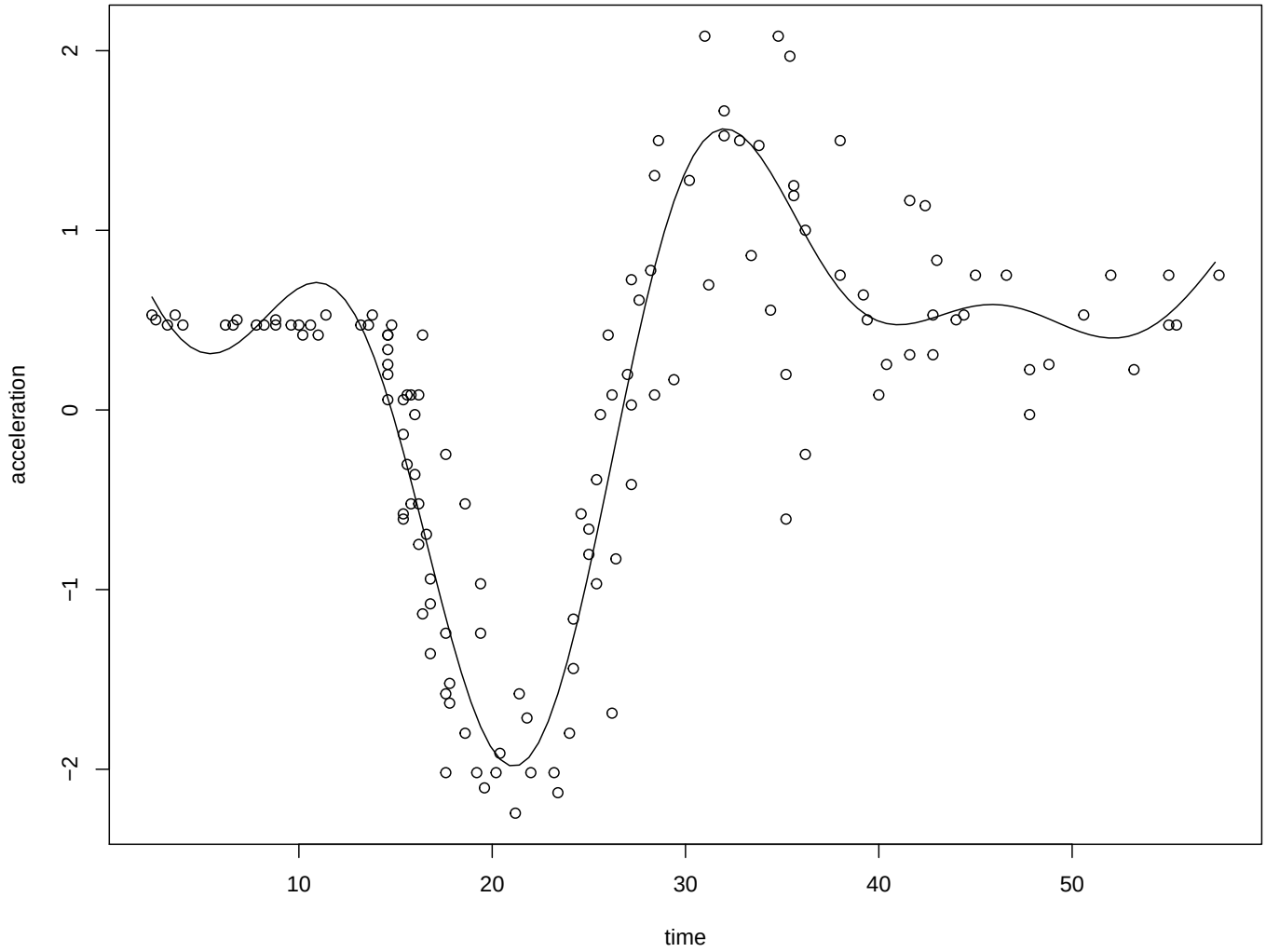
  in $\mathcal{H}$.
- Additional kernel parameters may be determined by cross-validation.

Overview

Application in R

```
library('kernlab')
library('MASS')
mcycle.data<-data.frame(mcycle)
mcycle.data$accel<-scale(mcycle.data$accel)
t<- seq(min(mcycle.data$times),max(mcycle.data$times),0.5)
testdat<-data.frame(times=t)

svmfit<-ksvm(accel~.,data=mcycle.data,scaled=TRUE,type="eps-svr",
             kernel="rbfdot",C=100,epsilon=0.1,kpar=list(sigma=1))

yhat<-predict(svmfit,newdata=testdat)
plot(mcycle.data$times,mcycle.data$accel)
lines(t,yhat)
```

Result

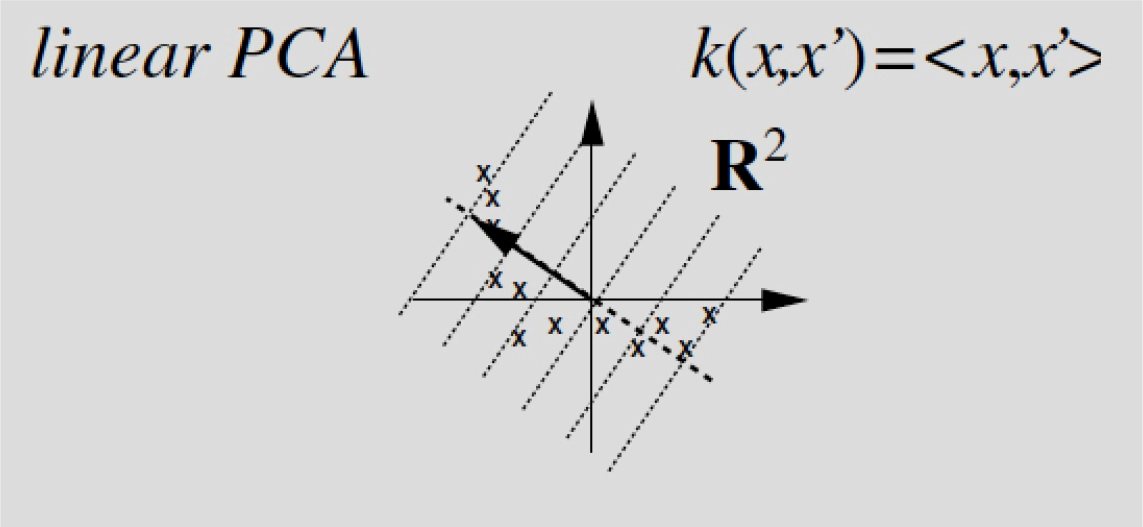


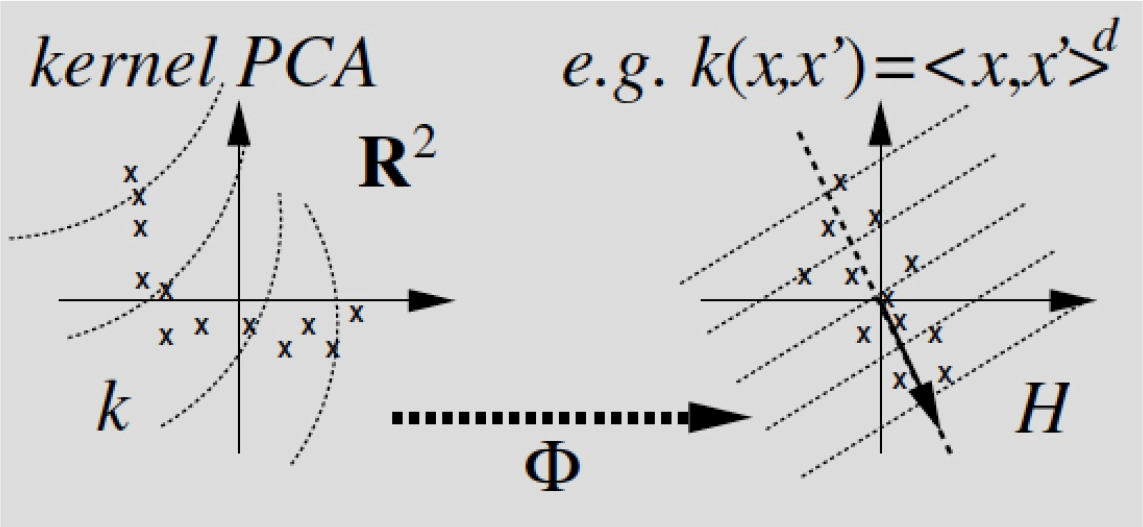
Overview

Kernel-based feature extraction

- The idea of implicitly mapping the data into a high-dimensional feature space has been very fruitful in the context of supervised leaning (SVM and SVR). Thus, it is natural to ask whether the same idea might prove useful in other domains of learning.
- The present section describes a kernel-based method for performing a nonlinear form of Principal Component Analysis, called Kernel PCA.
- We show that through the use of positive definite kernels, we can efficiently compute principal components in high-dimensional feature spaces, which are related to the input space by some nonlinear map.

Principle of KPCA





Overview

PCA (reminder)

- Principal Component Analysis (PCA) is a powerful technique for extracting structure from possibly high-dimensional data sets. It is readily performed by solving an eigenvalue problem.
- PCA is an orthogonal transformation (= transformation that preserves lengths of vectors and angles between them) of the coordinate system in which we describe our data.
- The new coordinate system is obtained by projection onto the so-called principal axes of the data. The new variables are called principal components or features.
- A small number of principal components is often sufficient to account for most of the structure in the data.

Covariance matrix

- Given a set of observations $\{x_i\}_{i=1}^n$, $x_i \in \mathbb{R}^p$, which are centered,

$$\sum_{i=1}^{n} x_i = 0,$$

PCA finds the principal axes by diagonalizing the sample covariance matrix,

$$\mathbf{C} = \frac{1}{n} \sum_{i=1}^{n} x_i x_i^T. \tag{2}$$

- Matrix $\mathbf{C}$ is positive definite, and can thus be diagonalized with nonnegative eigenvalues.

Diagonalizing $\mathbf{C}$

- To diagonalize $\mathbf{C}$, we solve the eigenvalue equation,

$$\lambda u = \mathbf{C}u$$

for eigenvalues $\lambda \geq 0$ and nonzero eigenvectors $u \in \mathbb{R}^p \setminus \{0\}$.

- Substituting (2) into this expression,

$$\lambda u = \mathbf{C}u = \frac{1}{n}\sum_{i=1}^{n} x_i x_i^T u = \frac{1}{n}\sum_{i=1}^{n}(x_i^T u)x_i$$

- Hence, every eigenvector u with $\lambda \neq 0$ can be written as some linear combination of the data vectors x_i (it lies in the span of $x_1, \ldots, x_n$).
- Instead of the eigenvalue equation $\lambda u = \mathbf{C}u$ we may consider the n projected equations

$$\lambda x_i^T u = x_i^T \mathbf{C}u, \quad i = 1, \ldots, n.$$

Overview

Feature space

- We now study PCA in the case where we are not interested in principal components in input space, but rather principal components of features that are nonlinearly related to the input variables.
- Let us consider a feature space $\mathcal{H}$, related to the input domain $\mathcal{X}$ (for instance, $\mathbb{R}^p$) by a map $\Phi : \mathcal{X} \to \mathcal{H}$, which is possibly nonlinear.
- The feature space could have an arbitrarily large, and possibly infinite, dimension.

Covariance matrix in $\mathcal{H}$

- Again, we assume that we are dealing with centered data,

$$\sum_{i=1}^{n} \Phi(x_i) = 0.$$

- In $\mathcal{H}$, the covariance matrix takes the form

$$\mathbf{C} = \frac{1}{n} \sum_{j=1}^{n} \Phi(x_j)\Phi(x_j)^T.$$

(If $\mathcal{H}$ is infinite-dimensional, we may think of $\Phi(x_j)\Phi(x_j)^T$ as a linear operator on $\mathcal{H}$, mapping $x \mapsto \Phi(x_j)\langle \Phi(x_j), x \rangle$).

Eigenvalue problem in $\mathcal{H}$

- We now have to find eigenvalues $\lambda \geq 0$ and nonzero eigenvectors $u \in \mathcal{H} \setminus \{0\}$ satisfying

$$\lambda u = \mathbf{C}u.$$

- Again, all solutions u with $\lambda \neq 0$ lie in the span of $\Phi(x_1), \ldots, \Phi(x_n)$. Consequently,
  1. We may instead consider the set of equations

$$\lambda \langle \Phi(x_k), u \rangle = \langle \Phi(x_k), \mathbf{C}u \rangle, \quad k = 1, \ldots, n, \tag{3}$$

  2. There exist coefficients α_i, $i = 1, \ldots, n$ such that

$$u = \sum_{i=1}^{n} \alpha_i \Phi(x_i). \tag{4}$$

Dual eigenvector representation

Using the bilinearity of the dot product and combining (3) and (4), we get

$$\lambda \sum_{i=1}^{n} \alpha_i \langle \Phi(x_k), \Phi(x_i) \rangle = \left\langle \Phi(x_k), \sum_{i=1}^{n} \alpha_i \mathbf{C} \Phi(x_i) \right\rangle \tag{5a}$$

$$= \frac{1}{n} \sum_{i=1}^{n} \alpha_i \left\langle \Phi(x_k), \sum_{j=1}^{n} \Phi(x_j) \langle \Phi(x_j), \Phi(x_i) \rangle \right\rangle \tag{5b}$$

for $k = 1, \ldots, n$.

Dual eigenvector representation (continued)

- Let $\mathbf{K}$ be the $n \times n$ Gram matrix with general term $K_{ij} = \langle \Phi(x_i), \Phi(x_j) \rangle$.
- Eqs (5a) can be written as

$$n\lambda \mathbf{K}\alpha = \mathbf{K}^2\alpha. \tag{6}$$

- Proof:

$$(\mathbf{K}\alpha)_k = \sum_i K_{ki}\alpha_i = \sum_i \alpha_i \langle \Phi(x_k), \Phi(x_i) \rangle$$

$$(\mathbf{K}^2\alpha)_k = \sum_i (K^2)_{ki}\alpha_i = \sum_i \alpha_i \sum_j K_{kj}K_{ji}$$

$$= \sum_i \alpha_i \sum_j \langle \Phi(x_k), \Phi(x_j) \rangle \langle \Phi(x_j), \Phi(x_i) \rangle$$

$$= \sum_{i=1}^{n} \alpha_i \left\langle \Phi(x_k), \sum_j \Phi(x_j) \langle \Phi(x_j), \Phi(x_i) \rangle \right\rangle$$

Dual eigenvector representation (continued)

- To find the solutions of (6) we solve the eigenvalue problem

$$n\lambda\alpha = \mathbf{K}\alpha \tag{7}$$

- Let $\lambda_1 \geq \ldots \geq \lambda_n$ denote the eigenvalues of $\mathbf{K}$ (the values $n\lambda$ in (7)), and $\alpha^1, \ldots, \alpha^n$ the corresponding eigenvectors.
- Let λ_q be the last nonzero eigenvalue.

Normalization of eigenvectors

- We normalize $\alpha^1, \ldots, \alpha^q$ by requiring that the corresponding vectors in $\mathcal{H}$ be normalized:

$$\langle u^m, u^m \rangle = \|u^m\|^2 = 1, \quad m = 1, \ldots, q.$$

- Using (4) and (7), this translates to

$$1 = \left\langle \sum_{i=1}^{n} \alpha_i^m \Phi(x_i), \sum_{j=1}^{n} \alpha_j^m \Phi(x_j) \right\rangle = \sum_{i,j} \alpha_i^m \alpha_j^m \langle \Phi(x_i), \Phi(x_j) \rangle$$
$$= \sum_{i,j} \alpha_i^m \alpha_j^m K_{ij} = \langle \alpha^m, \mathbf{K}\alpha^m \rangle = \lambda_m \langle \alpha^m, \alpha^m \rangle$$

- We thus normalize the α^m by

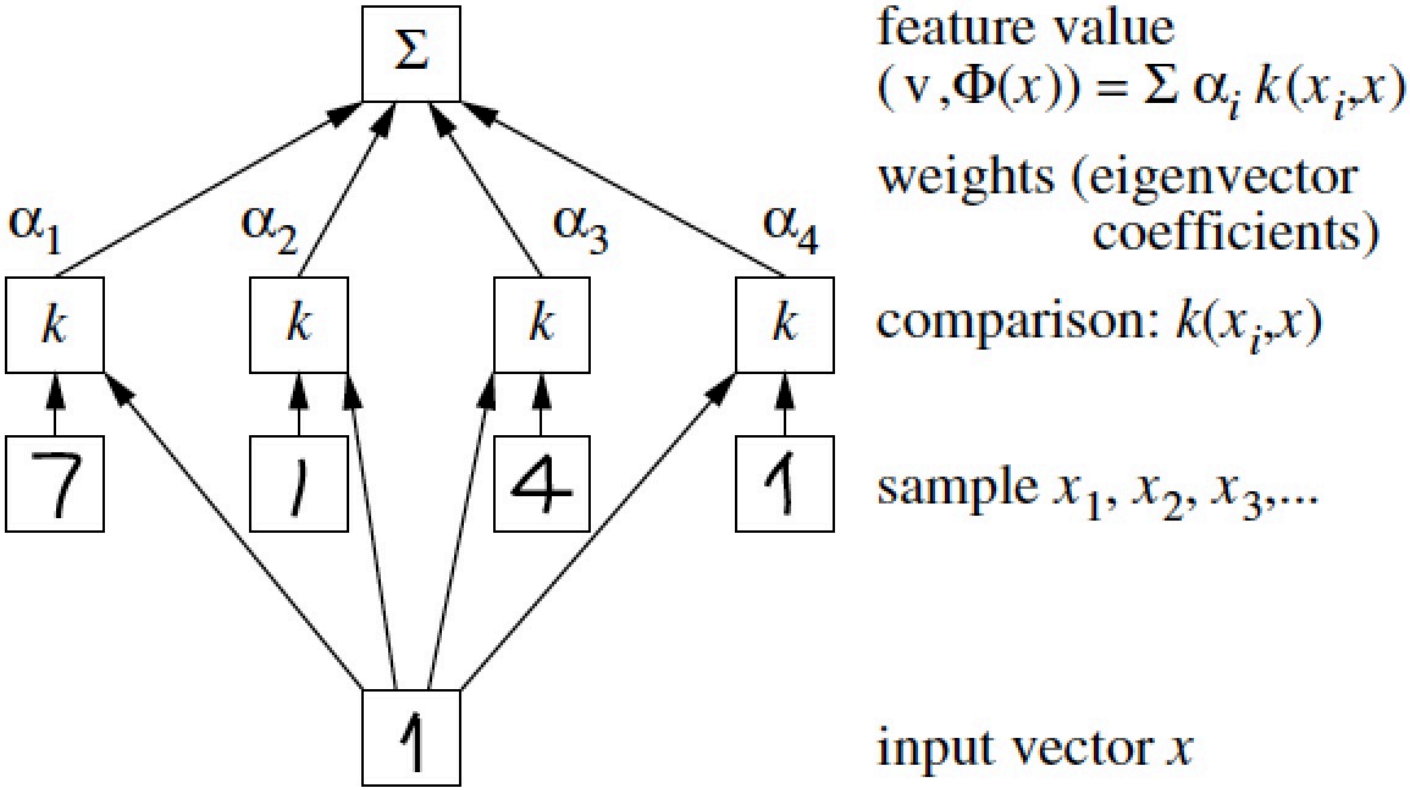
$$\alpha^m \leftarrow \frac{\alpha^m}{\|\alpha^m\| \sqrt{\lambda_m}}$$

Feature extraction

- For feature extraction, we need to compute the projections onto the eigenvectors u^m in $\mathcal{H}$.
- Let x be a test vector. The m-th feature is

$$z_m = \langle u^m, \Phi(x) \rangle = \left\langle \sum_{i=1}^{n} \alpha_i^m \Phi(x_i), \Phi(x) \right\rangle$$
$$= \sum_{i=1}^{n} \alpha_i^m \langle \Phi(x_i), \Phi(x) \rangle = \sum_{i=1}^{n} \alpha_i^m \mathcal{K}(x_i, x).$$

Feature extractor constructed using Kernel PCA



In the first layer, the input vector is compared to the sample via a kernel function, chosen a priori (e.g. polynomial, Gaussian, or sigmoid). The outputs are then linearly combined using weights, which are found by solving an eigenvector problem

Summary of KPCA

1. Compute the Gram matrix $\mathbf{K} = (\mathcal{K}(x_i, x_j))$.
2. Diagonalize $\mathbf{K}$, and normalize the eigenvector expansion coefficients α^m by requiring $\lambda_m \langle \alpha^m, \alpha^m \rangle = 1$.
3. To extract the principal components of a test point x, compute projections onto the eigenvectors by

$$z_m = \sum_{i=1}^{n} \alpha_i^m \mathcal{K}(x_i, x), \quad m = 1, \ldots, q.$$

Centering

- Until now, we have made the assumption that the observations are centered. This is easy to achieve in input space, but more difficult in $\mathcal{H}$, as we cannot explicitly compute the mean of the mapped observations.
- There is a way to do it, however, and this leads to slightly modified equations for kernel PCA.

Centering (continued)

- Let us assume that the $\Phi(x_i)$ are not centered.
- The Gram matrix in terms of the centered features is then

$$\widetilde{K}_{ij} = \left\langle \Phi(x_i) - \frac{1}{n}\sum_{k=1}^{n}\Phi(x_k), \Phi(x_j) - \frac{1}{n}\sum_{\ell=1}^{n}\Phi(x_\ell) \right\rangle$$

- It can be written as

$$\widetilde{\mathbf{K}} = \mathbf{K} - \mathbf{1}_n\mathbf{K} - \mathbf{K}\mathbf{1}_n + \mathbf{1}_n\mathbf{K}\mathbf{1}_n,$$

where $\mathbf{1}_n$ is the $n \times n$ matrix with general term $(\mathbf{1}_n)_{ij} = 1/n$.

Centering (continued)

To project a new test vector, we compute

$$\left\langle u^m, \Phi(x) - \frac{1}{n}\sum_{j=1}^{n}\Phi(x_j)\right\rangle =$$

$$\sum_{i=1}^{n}\alpha_i^m\left\langle \Phi(x_i) - \frac{1}{n}\sum_{j=1}^{n}\Phi(x_j), \Phi(x) - \frac{1}{n}\sum_{j=1}^{n}\Phi(x_j)\right\rangle = \sum_{i=1}^{n}\alpha_i^m\widetilde{\mathcal{K}}(x_i, x)$$

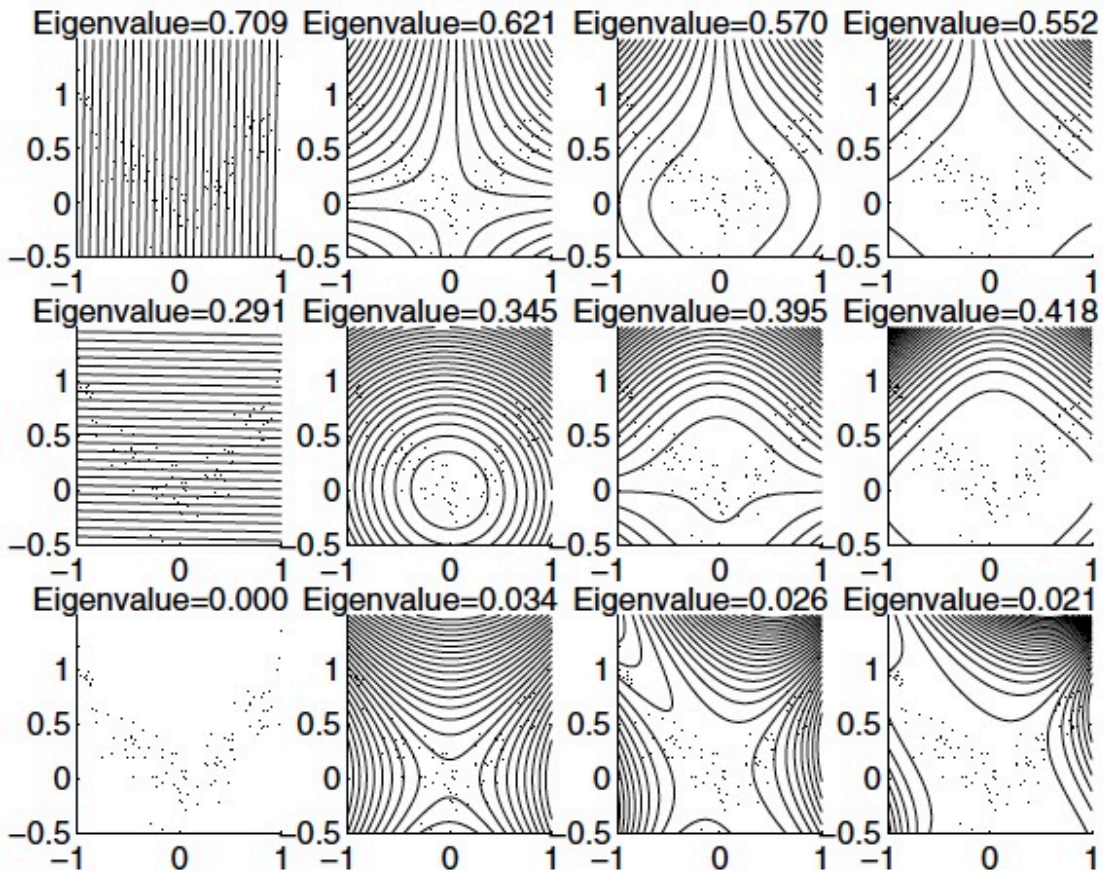
with

$$\widetilde{\mathcal{K}}(x_i, x) = \mathcal{K}(x_i, x) - \frac{1}{n}\sum_{k=1}^{n}\mathcal{K}(x_k, x)$$

$$- \frac{1}{n}\sum_{k=1}^{n}\mathcal{K}(x_i, x_k) + \frac{1}{n^2}\sum_{k,\ell}^{n}\mathcal{K}(x_k, x_\ell)$$

Overview

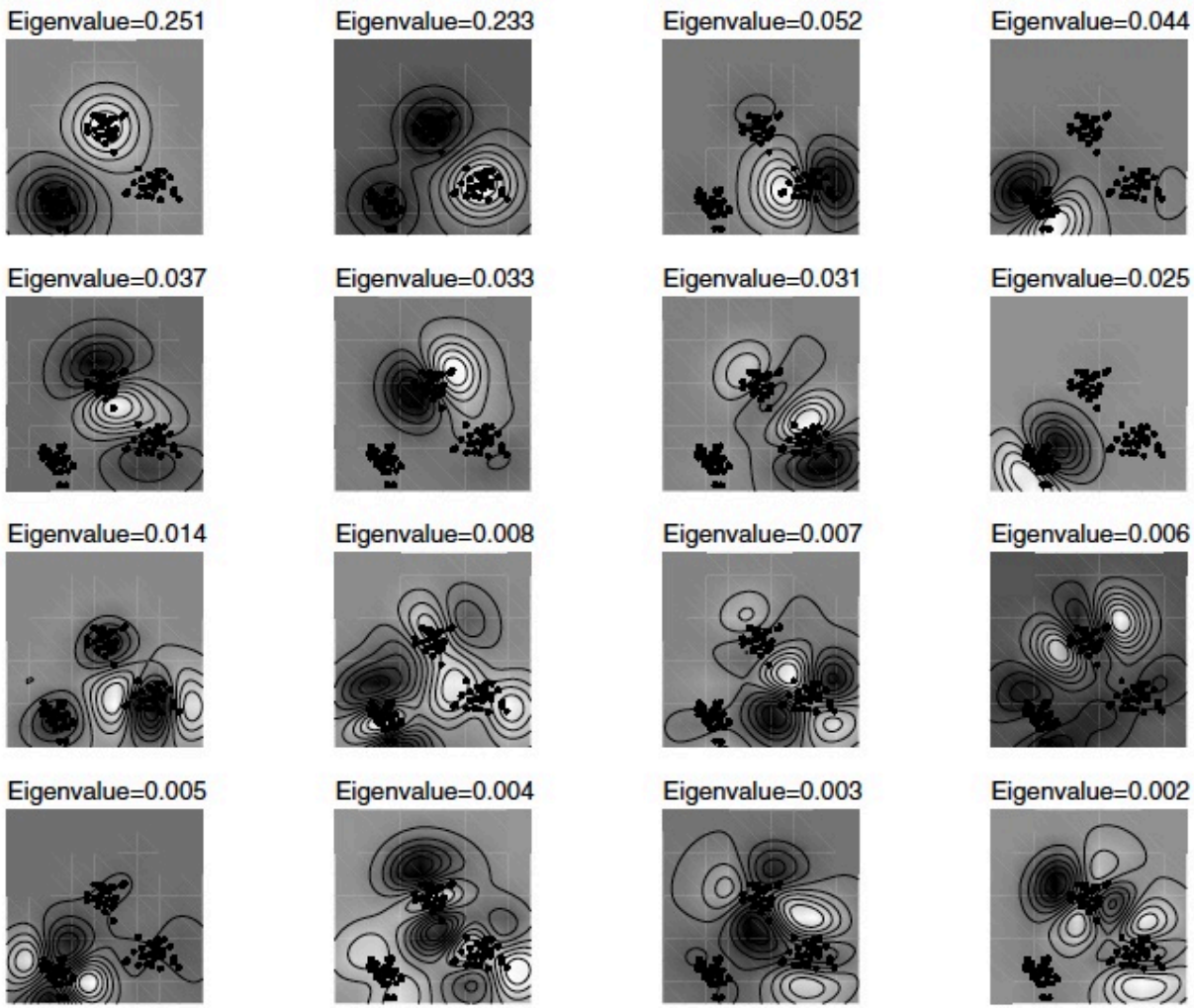
Two-dimensional toy example with polynomial kernel



From left to right, the polynomial degree in the kernel increases from 1 to 4; from top to bottom, the first 3 eigenvectors are shown, in order of decreasing eigenvalue size (eigenvalues are normalized to sum to 1).

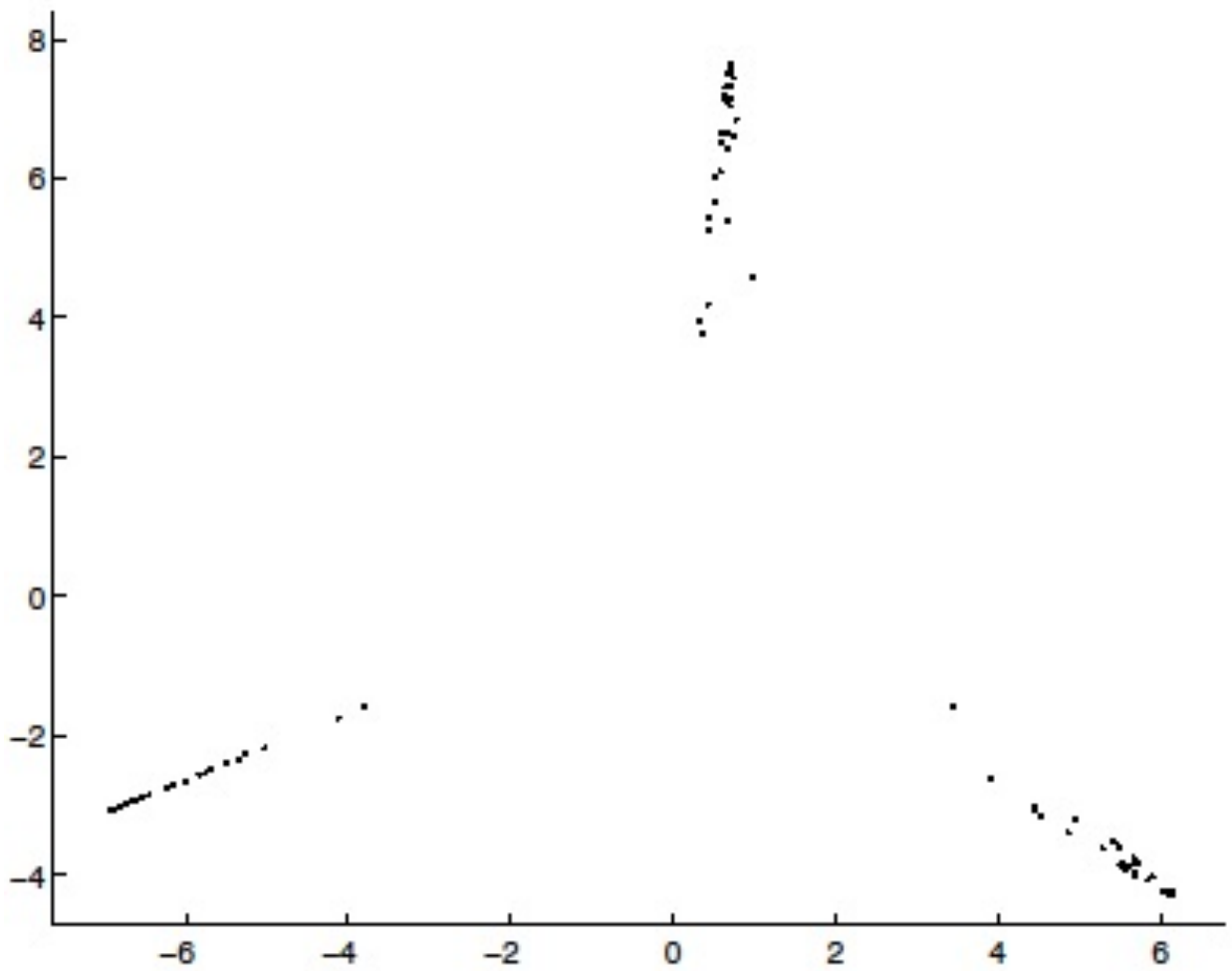
Example with 3 clusters and radial kernel ($\gamma = 0.1$)

First 16 nonlinear principal components



Example with 3 clusters and radial kernel ($\gamma = 0.1$)

Plot in the space of the first two principal components



Handwritten character recognition

- US postal service database: 9298 examples of dimensionality 256, of which 2007 make up the test set.
- For computational reasons, a subset of 3000 training examples was used to compute the matrix $\mathbf{K}$.
- Polynomial Kernel PCA was then used to extract nonlinear principal components from the training and test set.
- To assess the utility of the components (or features), a linear SVM was trained on the classification task.

Result

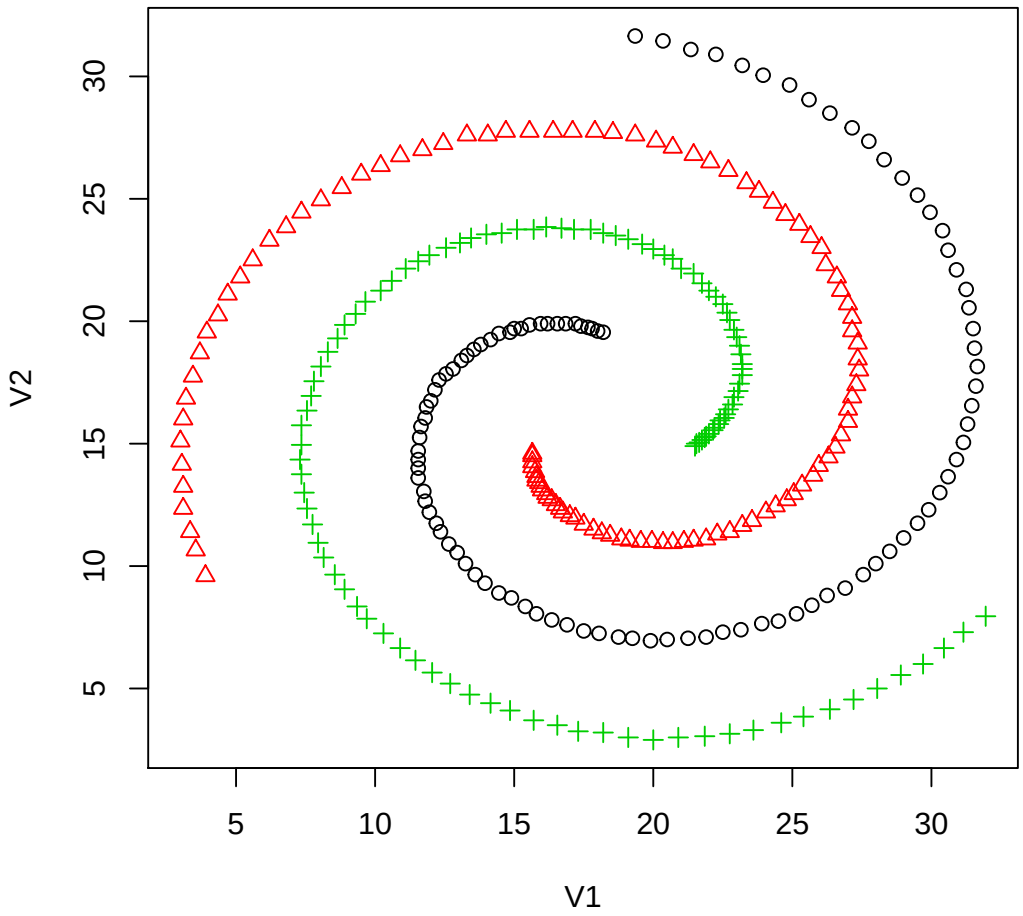
# of components	Test Error Rate for degree						
	1	2	3	4	5	6	7
32	9.6	8.8	8.1	8.5	9.1	9.3	10.8
64	8.8	7.3	6.8	6.7	6.7	7.2	7.5
128	8.6	5.8	5.9	6.1	5.8	6.0	6.8
256	8.7	5.5	5.3	5.2	5.2	5.4	5.4
512	n.a.	4.9	4.6	4.4	5.1	4.6	4.9
1024	n.a.	4.9	4.3	4.4	4.6	4.8	4.6
2048	n.a.	4.9	4.2	4.1	4.0	4.3	4.4

The case of degree 1 corresponds to standard PCA, with the number of nonzero eigenvalues being at most the dimensionality of the space (256). Clearly, nonlinear principal components afford test error rates which are lower than in the linear case (degree 1).

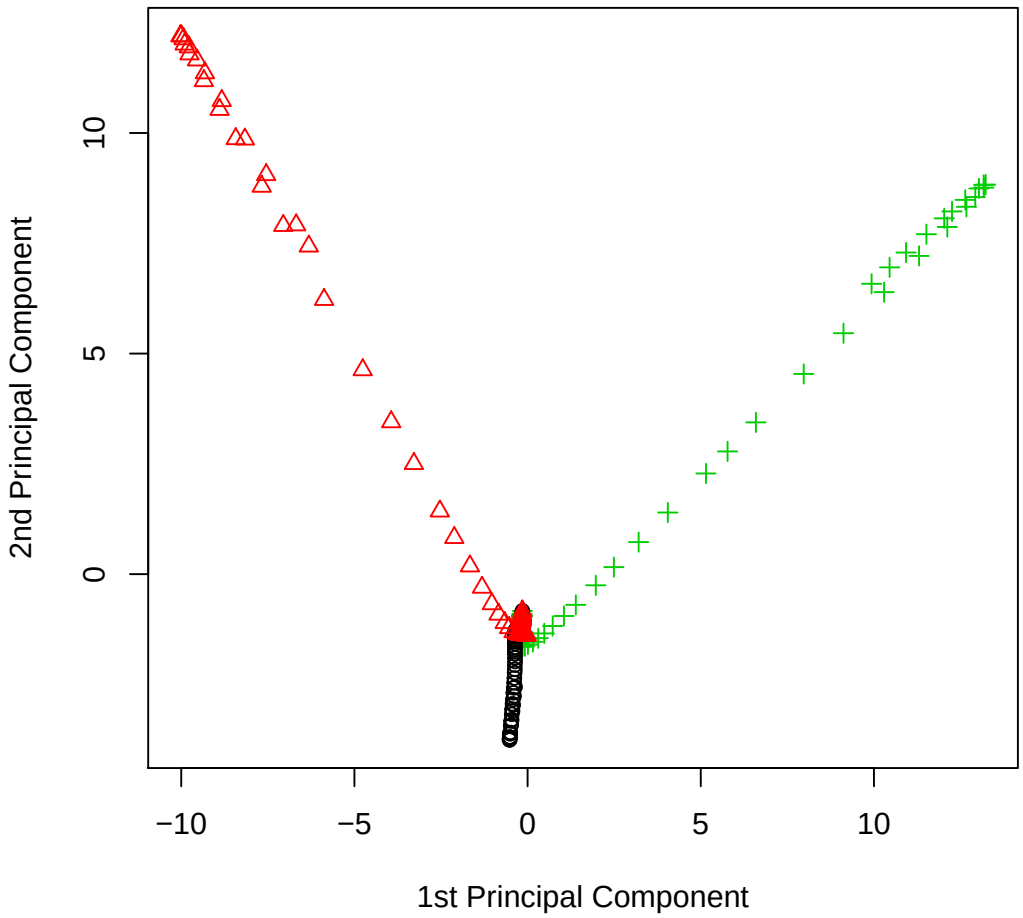
KPCA in R

```
library(kernlab)

spiral<-read.table(file='spiral.txt')
x<-as.matrix(spiral[,1:2])
y<-spiral[,3]
plot(x,col=y,pch=y)

kpc <- kpca(x,kernel="rbfdot",kpar=list(sigma=0.3))
plot(rotated(kpc)[,1:2,col=y,pch=y,xlab="1st Principal Component",
ylab="2nd Principal Component")
```

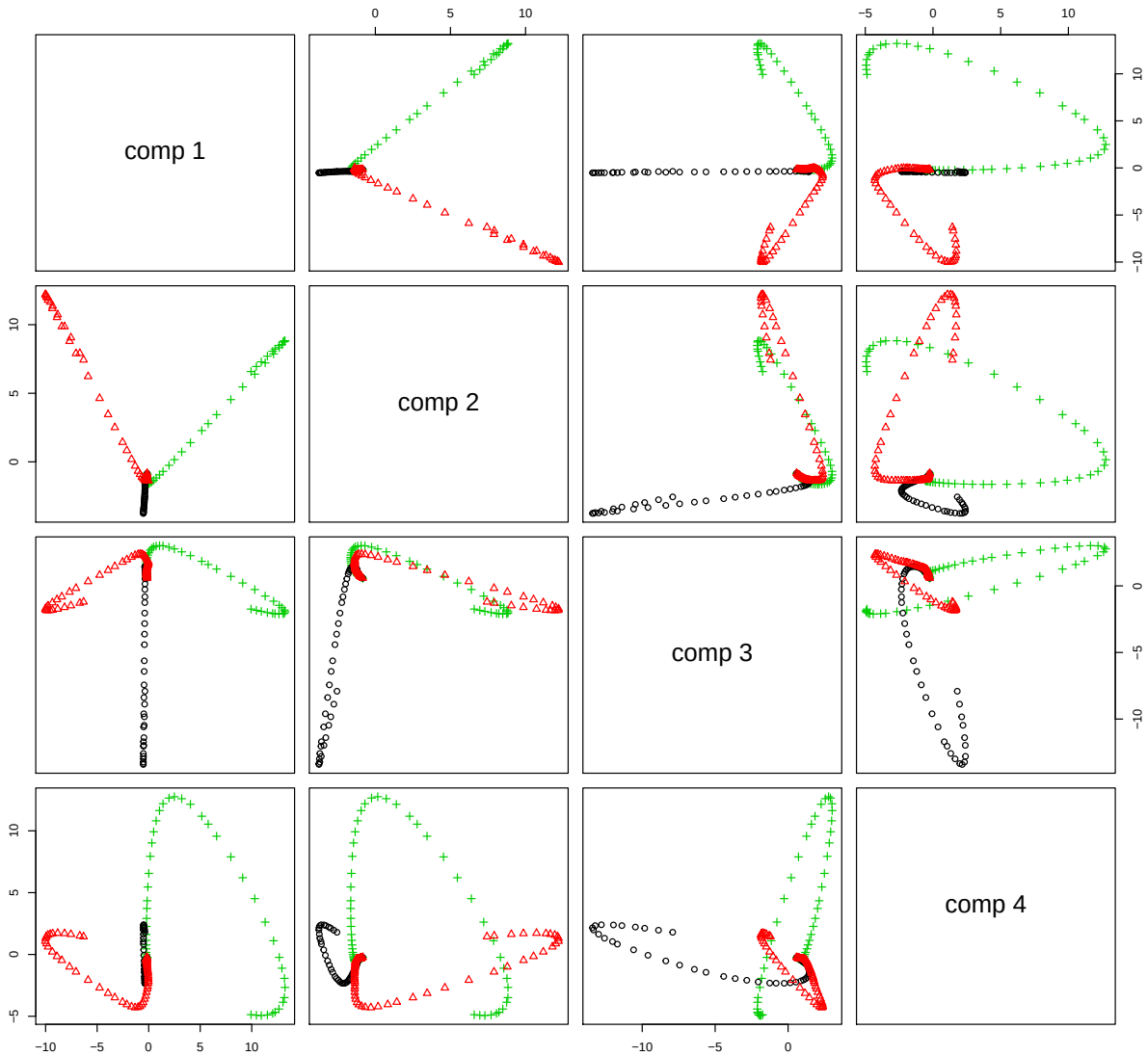
Spirals data



First two principal components



First four principal components



Cumulated variance