# SY19 – Machine Learning

## Chapter 9: Kernel-based methods for unsupervised learning

Thierry Denœux

Université de technologie de Compiègne

https://www.hds.utc.fr/~tdenoeux
email: tdenoeux@utc.fr

Automne 2024



## 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 or regression 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 the application of this principle to three unsupervised learning problems:
  1. Novelty detection: one-class SVM
  2. Clustering: kernel $c$-means
  3. Feature extraction: kernel PCA



## Overview

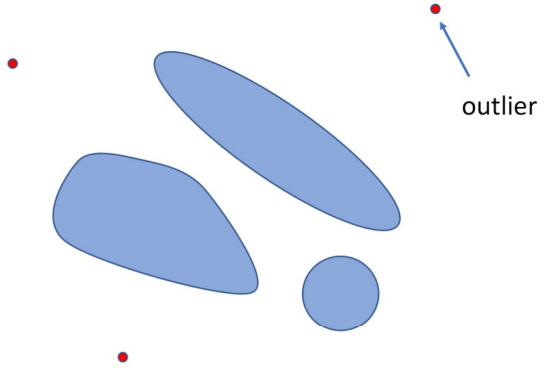




## Novelty detection

**Definition**

Novelty detection (also called anomaly or outlier detection) is the task of identifying observations that differ from the rest of the data, and are thus likely to be drawn from a different distribution.



## Applications of novelty detection

- Applications:
  - Data cleaning (remove spurious observations corresponding, e.g., to measurement errors)
  - Technical diagnosis, process monitoring (detect new states of the machine/process that are not represented in the learning set)
  - Detection of abnormal events:
    - Bank fraud detection (detect abnormal activity related to an account)
    - Network intrusion detection
  - Etc.
- Main approaches:
  1. Density estimation
  2. Minimum-volume region estimation





## Overview







## Density estimation

- A classical approach is to compute an estimate $\widehat{f}$ of the density function $f$ and to reject the input vector $x$ if $\widehat{f}(x) < c$ for some constant $c$.
- The threshold $c$ can be tuned to reject some small proportion of the learning vectors.
- Density estimation methods:
  - Parametric: based on a parametric statistical model (e.g., Gaussian distribution or Gaussian mixture)
  - Nonparametric: Parzen window (kernel) density estimation





## Kernel density estimation

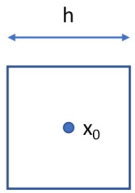
- Suppose we have a random sample of $n$ $p$-dimensional vectors $x_1, \ldots, x_n$ drawn from a probability density $f(x)$, and we wish to estimate $f$ at some point $x_0$.
- Consider a small region $\mathcal{R}$ centered at $x_0$, with volume $V$. If $\mathcal{R}$ is small enough, $f$ is nearly constant in $\mathcal{R}$, so that

$$P(X \in \mathcal{R}) = \int_{\mathcal{R}} f(x)dx \approx f(x_0) \int_{\mathcal{R}} dx = f(x_0) \times V$$

- Now, $P(X \in \mathcal{R})$ can be estimated by $\frac{1}{n}\sum_{i=1}^{n} I(x_i \in \mathcal{R})$, so $f(x_0)$ can be estimated by

$$\boxed{\widehat{f}(x_0) = \frac{1}{nV}\sum_{i=1}^{n} I(x_i \in \mathcal{R})}$$

## Kernel density estimation (continued)




- If $\mathcal{R}$ is a hypercube centered at $x_0$ with side length $h$, then $\widehat{f}(x_0)$ can be written as

$$\widehat{f}(x_0) = \frac{1}{nh^p} \sum_{i=1}^{n} H\left(\frac{x_i - x_0}{h}\right)$$

where $H$ is the uniform density on the unit hypercube:

$$H(u) = I\left(|u_j| \leq 1/2, \forall j \in \{1, \ldots, p\}\right)$$

- This function is piecewise constant, so it is not continuous.





## Kernel density estimation (continued)

- To obtain a smoother density estimate, we can replace the uniform density by a smoother density centered at 0, which decreases with the distance to the origin, such as the standard Gaussian density:

$$\phi(u) = \frac{1}{(2\pi)^{p/2}} \exp\left(-\frac{1}{2} u^T u\right)$$

- We then have

$$\widehat{f}(x_0) = \frac{1}{n} \sum_{i=1}^{n} \underbrace{\frac{1}{h^p} \phi\left(\frac{x_i - x_0}{h}\right)}_{\mathcal{K}_h(x_i, x_0)} \tag{1}$$

where $\mathcal{K}_h$ is the kernel function.

- Parameter $h$ is called the bandwidth of the kernel. It controls the smoothness of the density estimate.





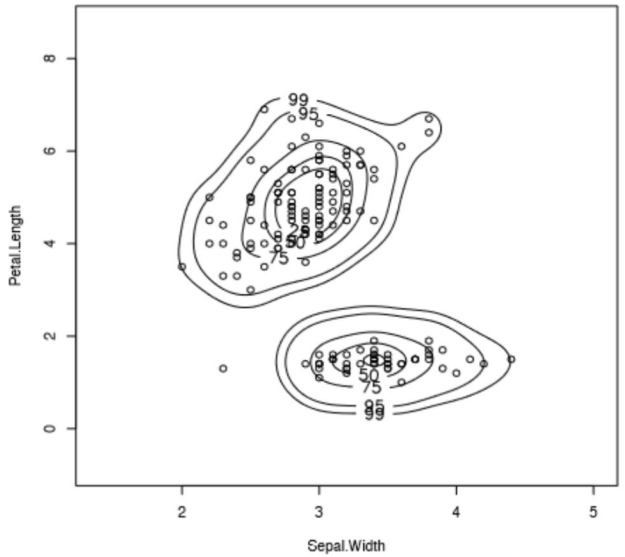
## Kernel density estimation in R

```
library('ks')
data(iris)
x<-iris[,c(2,3)]
fhat <- kde(x)
plot(fhat,cont=c(25,50,75,95,99),labcex=1.2)
points(x)
```





## Result



## Minimum volume estimator

- Density estimation is increasingly difficult as the dimension $p$ increases. This is particularly true for nonparametric density estimation.
- Actually, we do not need to estimate the precise value of $f$ at any $x$, but we only need to find a minimum volume region $\mathcal{R}$ such that $P(X \in \mathcal{R}) \geq \mu$ for some $\mu \in (0, 1)$.
- Vapnik's principle: when solving a problem of interest, do not solve a more general problem as an intermediate step.
- Here, we will search for a minimal volume hypersphere that contains a fraction of the data in a transformed feature space.





## Overview







## Primal optimization problem

- We consider a mapping $\Phi : \mathbb{R}^p \to \mathcal{H}$ defined by a kernel $\mathcal{K}$, i.e.,

$$\langle \Phi(x), \Phi(x') \rangle = \mathcal{K}(x, x')$$

- We want to find a minimum-volume ball with center $c$ and radius $R$ in $\mathcal{H}$ containing most of the data. This can be formalized as follows:

$$\min_{R,c,\xi} R^2 + \frac{1}{\nu n} \sum_{i=1}^{n} \xi_i \tag{2}$$

subject to

$$\|\Phi(x_i) - c\|^2 \leq R^2 + \xi_i, \quad i = 1, \ldots, n \tag{3a}$$

$$\xi_i \geq 0, \quad i = 1, \ldots, n \tag{3b}$$

- Here, $\nu$ is a coefficient whose meaning will be clarified later.





## Lagrange function

- We can rewrite the squared distances using dot products as

$$\begin{aligned} \|\Phi(x_i) - c\|^2 &= \langle \Phi(x_i) - c, \Phi(x_i) - c \rangle \\ &= \langle \Phi(x_i), \Phi(x_i) \rangle - 2\langle \Phi(x_i), c \rangle + \langle c, c \rangle \end{aligned}$$

- Consequently, the Lagrange function can be written as

$$L(R, c, \xi, \alpha, \beta) = R^2 + \frac{1}{\nu n} \sum_{i=1}^{n} \xi_i -$$

$$\sum_{i=1}^{n} \alpha_i \left( -\langle \Phi(x_i), \Phi(x_i) \rangle + 2\langle \Phi(x_i), c \rangle - \langle c, c \rangle + R^2 + \xi_i \right) -$$

$$\sum_{i=1}^{n} \beta_i \xi_i \tag{4}$$

## Derivatives of the Lagrange function

Setting the derivatives of the Lagrange function to zero, we get

$$\frac{\partial L}{\partial R} = 2R - 2R\sum_i \alpha_i = 0 \Rightarrow \boxed{\sum_i \alpha_i = 1} \tag{5}$$

$$\frac{\partial L}{\partial \xi_i} = \frac{1}{\nu n} - \alpha_i - \beta_i = 0 \Rightarrow \boxed{\alpha_i = \frac{1}{\nu n} - \beta_i} \tag{6}$$

from which we deduce

$$\boxed{0 \leq \alpha_i \leq \frac{1}{\nu n}} \tag{7}$$

and

$$\frac{\partial L}{\partial c} = -2\sum_i \alpha_i \Phi(x_i) + 2c \underbrace{\sum_i \alpha_i}_{1} = 0 \Rightarrow \boxed{c = \sum_i \alpha_i \Phi(x_i)} \tag{8}$$





## Simplification of the Lagrange function

Substituting (5)-(8) in (4), we get

$$L = R^2 + \sum_{i=1}^n \xi_i \underbrace{\left(\frac{1}{\nu n} - \alpha_i - \beta_i\right)}_{0} - R^2 \underbrace{\sum_i \alpha_i}_{1} +$$

$$\sum_{i=1}^n \alpha_i \left(\langle \Phi(x_i), \Phi(x_i)\rangle - 2\langle \Phi(x_i), c\rangle + \langle c, c\rangle\right)$$

$$= \sum_{i=1}^n \alpha_i \langle \Phi(x_i), \Phi(x_i)\rangle - 2 \underbrace{\sum_i \alpha_i \left\langle \Phi(x_i), \sum_j \alpha_j \Phi(x_j)\right\rangle}_{\sum_{i,j} \alpha_i \alpha_j \langle \Phi(x_i), \Phi(x_j)\rangle} +$$

$$\underbrace{\left\langle \sum_i \alpha_i \Phi(x_i), \sum_j \alpha_j \Phi(x_j)\right\rangle}_{\sum_{i,j} \alpha_i \alpha_j \langle \Phi(x_i), \Phi(x_j)\rangle} \underbrace{\sum_i \alpha_i}_{1}$$





## Wolfe dual problem

- The Lagrange function of the dual problem is, thus,

$$L_D(\alpha) = \sum_{i=1}^n \alpha_i \mathcal{K}(x_i, x_i) - \sum_{i,j} \alpha_i \alpha_j \mathcal{K}(x_i, x_j)$$

- If $\mathcal{K}(x, x')$ depends only on $x - x'$ (as with RBF kernels), $\mathcal{K}(x, x)$ is a constant. The Wolfe dual problem can then be written as

$$\min_\alpha \sum_{i,j} \alpha_i \alpha_j \mathcal{K}(x_i, x_j)$$

subject to

$$\sum_{i=1}^n \alpha_i = 1 \quad \text{and} \quad 0 \leq \alpha_i \leq \frac{1}{\nu n}, \quad i = 1, \ldots, n$$





## Overview

## Support vectors

- We have the KKT conditions

$$\alpha_i(R^2 + \xi_i - \|\Phi(x_i) - c\|^2) = 0$$

$$\beta_i \xi_i = 0$$

- The support vectors ($\alpha_i > 0$) are such that $\|\Phi(x_i) - c\|^2 = R^2 + \xi_i$. They lie
  - Outside the ball ("outliers") if $\beta_i = 0$, which implies $\alpha_i = \frac{1}{\nu n}$, or
  - On the surface ("in-bound") if $\beta_i > 0$, which implies $0 < \alpha_i < \frac{1}{\nu n}$.
- The learning vectors inside the ball verify $\alpha_i = 0$.





## Decision function

- The decision function is

$$g(x) = \text{sign}\left(R^2 - \|\Phi(x) - c\|^2\right) \tag{9a}$$

$$= \text{sign}\left(R^2 - \mathcal{K}(x,x) + 2\mathcal{K}(x,c) - \mathcal{K}(c,c)\right) \tag{9b}$$

$$= \text{sign}\left(R^2 - \mathcal{K}(x,x) + 2\sum_{i\in\mathcal{S}} \alpha_i \mathcal{K}(x,x_i) - \sum_{(i,j)\in\mathcal{S}^2} \alpha_i\alpha_j \mathcal{K}(x_i,x_j)\right) \tag{9c}$$

  where $\mathcal{S}$ is the set of indices of SVs.
- $R^2$ can be computed from any in-bound SV $x_i$ as

$$R^2 = \|\Phi(x_i) - c\|^2$$

$$= \mathcal{K}(x_i,x_i) - 2\sum_{j\in\mathcal{S}} \alpha_j \mathcal{K}(x_i,x_j) + \sum_{(j,l)\in\mathcal{S}^2} \alpha_j\alpha_l \mathcal{K}(x_j,x_l)$$





## Role of $\nu$ and link with kernel density estimation

- When $\nu \to 0$, the penalization term in (2) tends to infinity except if all $\xi_i = 0$: the constraint $\|\Phi(x_i) - c\|^2 \leq R^2$ is enforced for all $x_i$. This is not a problem as $R$ can be large enough so that the ball contains all the data. (In this case there is no SV).
- When $\nu = 1$, from (5) and (7), we deduce $\alpha_i = 1/n$ for all $i$. Consequently, all the data points are SVs, and the decision function (9) becomes

$$g(x) = \text{sign}(\underbrace{\frac{1}{n}\sum_{i=1}^{n} \mathcal{K}(x,x_i)}_{\widehat{f}(x)} - \eta)$$

  i.e., it amounts to comparing the kernel density estimate (1) to a threshold $\eta$.





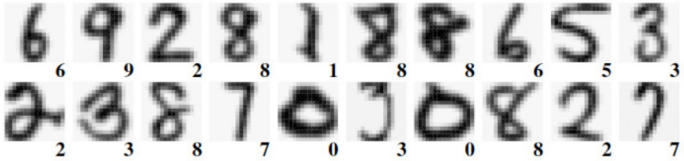
## Interpretation of $\nu$

**Proposition**

*Suppose we run a one-class SVM with a kernel $\mathcal{K}$ on some data. Then:*

1. *$\nu$ is an upper bound on the fraction of outliers (SVs with $\xi_i > 0$).*
2. *$\nu$ is a lower bound on the fraction of SVs.*
3. *Suppose the data $(x_1, y_1), \ldots, (x_n y_n)$ were generated iid from a continuous distribution. Under very general technical conditions, with probability 1, $\nu$ equals asymptotically both the fraction of SVs and the fraction of outliers.*

## Example with the USPS dataset

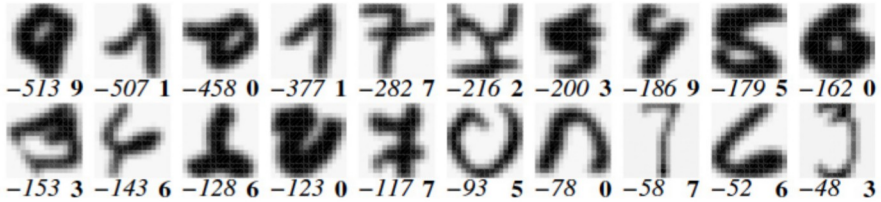


| $\nu$ | fraction of OLs | fraction of SVs | training time |
|---|---|---|---|
| 1% | 0.0% | 10.0% | 36 |
| 3% | 0.1% | 10.0% | 31 |
| 5% | 1.4% | 10.6% | 36 |
| 10% | 6.2% | 13.7% | 65 |
| 30% | 27.5% | 31.8% | 269 |
| 50% | 47.4% | 51.2% | 1284 |
| 70% | 68.3% | 70.7% | 1512 |
| 90% | 89.4% | 90.1% | 2349 |





## USPS dataset: examples of outliers

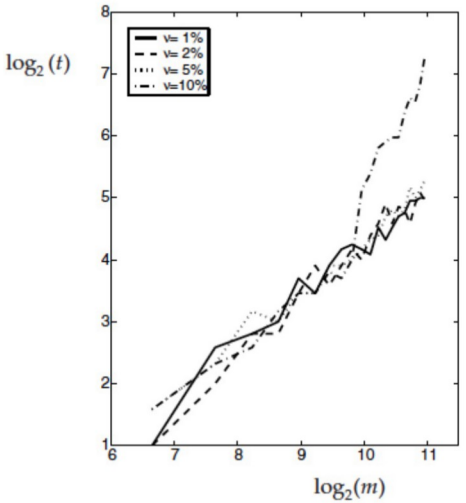



Outliers identified by the one-class SVM, ranked by the negative output.





## USPS dataset: training time




Training times $t$ (in seconds) vs. data set sizes $n$ (both axes depict logs at base 2; CPU time in seconds on a PC, training on subsets of the USPS test set). Larger values of $\nu$ generally lead to longer training times. For small values of $\nu$ ($\leq 5\%$), the training times are approximately linear in the training set. The scaling gets worse as $\nu$ increases.

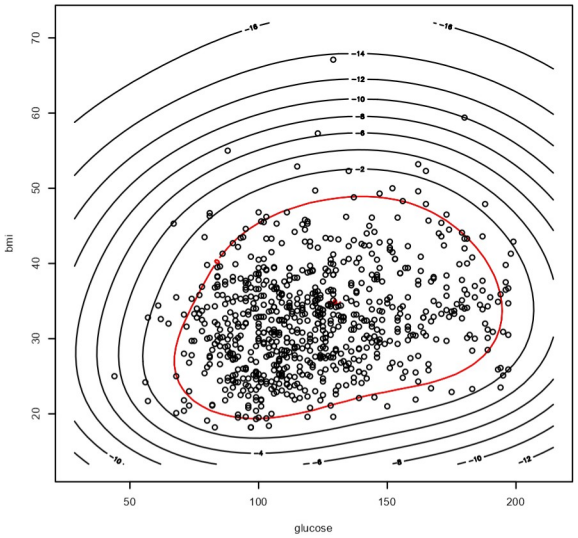




## One-class SVM in R

```
library('kernlab')
svmfit<-ksvm(~glucose+bmi,data=pima[ii,],type="one-svc",
             kernel="rbfdot",nu=0.1,kpar=list(sigma=0.3))
x<-as.matrix(pima[ii,c(2,6)])
nx<-50
ny<-50
Dx<-max(x[,1])-min(x[,1])
Dy<-max(x[,2])-min(x[,2])
xx<-seq(min(x[,1])-0.1*Dx,max(x[,1])+0.1*Dx,1.2*Dx/(nx-1))
yy<-seq(min(x[,2])-0.1*Dy,max(x[,2])+0.1*Dy,1.2*Dy/(ny-1))
z<-matrix(0,nx,ny)
for(i in 1:nx) for(j in 1:ny) z[i,j]<-predict(svmfit,
      newdata=data.frame(glucose=xx[i],bmi=yy[j]),type="decision")
contour(xx,yy,z,xlab="glucose",ylab="bmi")
contour(xx,yy,z,levels=0,lwd=2,col="red",add=TRUE)
points(x[,1],x[,2])
```

## Result








## Overview

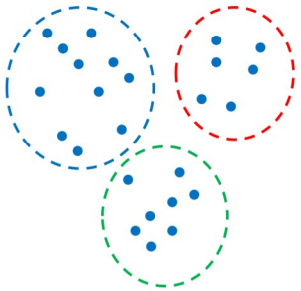






## Clustering (reminder)

- Clustering is a form of unsupervised learning that consists in finding groups (clusters) in data.
- In partitional clustering, we search for a partition of $n$ objects into $c$ groups, such that each object is assigned to one and only one group.
- Usually, the groups are defined in such a way that within-group distances are small relative to between-group distances.







## Formalization

- A partition into $c$ groups can be encoded as an $n \times c$ matrix $\mathbf{U} = (u_{ik})$ such that $u_{ik} = 1$ if vector $x_i$ belongs to group $k$, and $u_{ik} = 0$ otherwise.
- How to find a "good" matrix $\mathbf{U}$?
- We may want to minimize the sum of squared distances between each learning vector $x_i$ and the center of its cluster:

$$J(\mathbf{U}) = \sum_{k=1}^{c} \sum_{i=1}^{n} u_{ik} d_{ik}^2$$

where $d_{ik} = \|x_i - v_k\|$, and

$$v_k = \frac{\sum_{i=1}^{n} u_{ik} x_i}{\sum_{i=1}^{n} u_{ik}} \quad \text{is the center of cluster } k.$$

- This criterion can be minimized using the $c$-means algorithms.

## Overview







## $c$-Means algorithm

1. Fix the number $c$ of clusters
2. Initialize $c$ prototypes $v_1, \ldots, v_c$ randomly (can be taken from the input vectors $x_i$)
3. Assign each $x_i$ to the cluster with the nearest prototype
4. Recompute each prototype $v_k$ as the center of mass of class $k$:

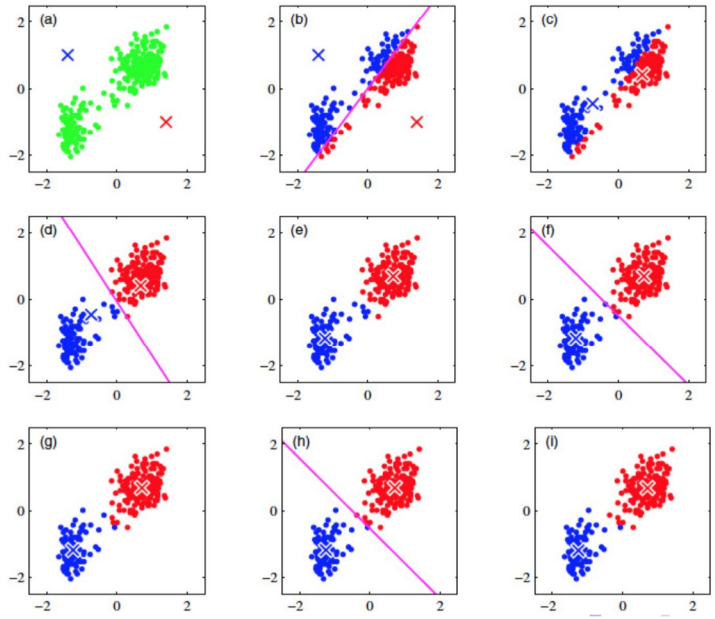
$$v_k \leftarrow \frac{\sum_{i=1}^{n} u_{ik} x_i}{\sum_{i=1}^{n} u_{ik}}, \quad k = 1, \ldots, c$$

5. If the prototypes have not changed in the last iteration, stop. Otherwise, return to Step 3.





## Illustration of the $c$-means algorithm



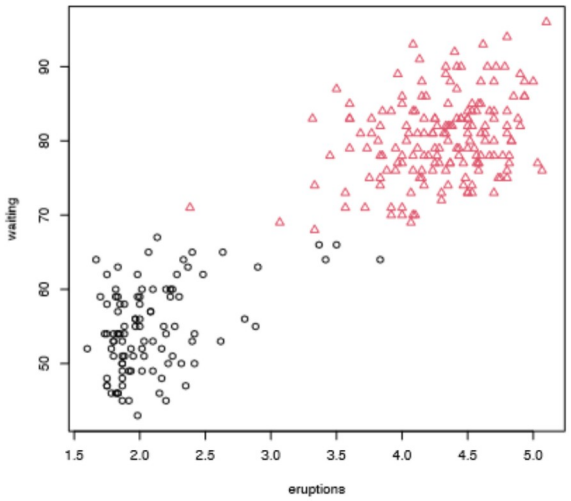





## Example in R

```
library(mclust)
data(faithful)

clus <- kmeans(faithful,c=2,nstart=10)

plot(faithful,col=km$cluster,pch=km$cluster)
```

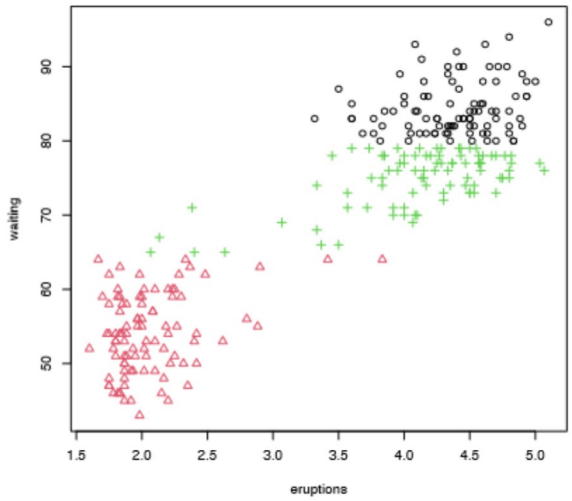
## Result (2 clusters)








## Result (3 clusters)








## Proof of convergence

- Let $\mathbf{V} = (v_1, \ldots, v_c)$ be a matrix containing the prototypes. Criterion $J(\mathbf{U})$ can be written as a function of $\mathbf{U}$ and $\mathbf{V}$:

$$\boxed{J(\mathbf{U}, \mathbf{V}) = \sum_{i=1}^{n} \sum_{k=1}^{c} u_{ik} d_{ik}^2}$$

- The algorithm alternates 2 steps:
  1. Update of $\mathbf{U}$ with $\mathbf{V}$ fixed
  2. Update of $\mathbf{V}$ with $\mathbf{U}$ fixed
- $J(\mathbf{U}, \mathbf{V})$ decreases at each step. As a consequence, the algorithm converges to a local minimum of $J(\mathbf{U}, \mathbf{V})$.





## Proof that $J(\mathbf{U}, \mathbf{V})$ decreases at each iteration

- Step 1: the cost function can be written as

$$J(\mathbf{U}, \mathbf{V}) = \sum_{i=1}^{n} \underbrace{\sum_{k=1}^{c} u_{ik} d_{ik}^2}_{d_{i,k(i)}^2} = \sum_{i=1}^{n} d_{i,k(i)}^2$$

When updating $\mathbf{U}$ for fixed $\mathbf{V}$, each $k(i)$ is chosen to minimize $d_{i,k(i)}^2$ (as $d_{i,k(i)} = \min_\ell d_{i\ell}$), and hence $J(\mathbf{U}, \mathbf{V})$.

## Proof that $J(\mathbf{U}, \mathbf{V})$ decreases at each iteration (cont.)

- Step 2: the cost function can alternatively be written as

$$J(\mathbf{U}, \mathbf{V}) = \sum_{k=1}^{c} \underbrace{\sum_{i=1}^{n} u_{ik} d_{ik}^2}_{\mathcal{I}(v_k)}$$

where

$$\mathcal{I}(v_k) = \sum_{i=1}^{n} u_{ik} (x_i - v_k)^T (x_i - v_k)$$

- We have

$$\frac{\partial \mathcal{I}(v_k)}{\partial v_k} = 0 \Leftrightarrow -2 \sum_{i=1}^{n} u_{ik} (x_i - v_k) = 0 \Leftrightarrow v_k = \frac{1}{n_k} \sum_{\{i : u_{ik} = 1\}} x_i$$

So, updating $\mathbf{V}$ in such a way that $v_k$ is the center of cluster $k$ minimizes $J(\mathbf{U}, \mathbf{V})$.





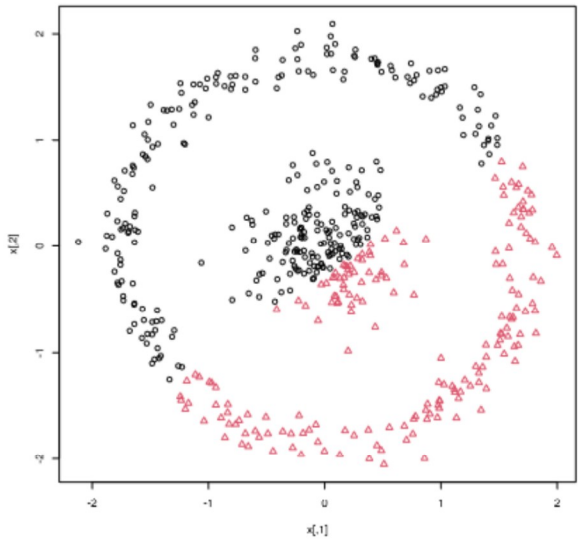
## Limitations of the $c$-means algorithm

- The $c$-means algorithm tends to find spherical clusters.
- We have seen that the EM algorithm applied to a GMM sometimes finds better partitions with ellipsoidal clusters.
- Yet, both $c$-means and EM may fail in the case of complex-shaped clusters with nonlinear boundaries.
- One solution is to use the kernel trick and find prototypes in a feature space $\mathcal{H}$ defined by a kernel $\mathcal{K}$.





## Example








## Overview



## Basic ideas

- As before, we consider a mapping $\Phi : \mathbb{R}^p \to \mathcal{H}$ defined by a kernel $\mathcal{K}$ through the equation

$$\langle \Phi(x), \Phi(x') \rangle = \mathcal{K}(x, x')$$

- We want to run the $c$-means algorithm in feature space $\mathcal{H}$, to minimize

$$J(\mathbf{U}) = \sum_{i=1}^{n} \sum_{k=1}^{c} u_{ik} \|\Phi(x_i) - v_k^{\Phi}\|^2$$

where

$$v_k^{\Phi} = \frac{\sum_{i=1}^{n} u_{ik} \Phi(x_i)}{\sum_{i=1}^{n} u_{ik}} \quad \text{is the center of cluster } k \text{ in feature space.}$$





## $c$-means in feature space

- The $c$-means in feature space would be as follows:
  1. Start with an initial partition $\mathbf{U}$ in $c$ clusters
  2. Recompute each prototype $v_k^{\Phi}$ as the center of mass of class $k$:

$$v_k^{\Phi} \leftarrow \frac{\sum_{i=1}^{n} u_{ik} \Phi(x_i)}{\sum_{i=1}^{n} u_{ik}}, \quad k = 1, \ldots, c$$

  3. Assign each $\Phi(x_i)$ to the cluster with the nearest prototype and update $\mathbf{U}$
  4. Return to Step 2 while the partition has changed.
- However, Step 2 cannot be performed directly because the mapping $\Phi$ is not known explicitly.
- We can compute distances in Step 3 without explicitly computing the prototypes using the kernel trick.





## Computing distances to prototypes

- We have $v_k^{\Phi} = \sum_{i=1}^{n} \gamma_{ik} \Phi(x_i)$, with $\gamma_{ik} = u_{ik} / \sum_{j=1}^{n} u_{jk}$.
- Hence,

$$\begin{aligned} \|\Phi(x_i) - v_k^{\Phi}\|^2 &= \langle \Phi(x_i) - \sum_{j=1}^{n} \gamma_{jk} \Phi(x_j), \Phi(x_i) - \sum_{j=1}^{n} \gamma_{jk} \Phi(x_j) \rangle \\ &= \mathcal{K}(x_i, x_i) - 2 \sum_{j=1}^{n} \gamma_{jk} \mathcal{K}(x_i, x_j) + \sum_{j,r} \gamma_{jk} \gamma_{rk} \mathcal{K}(x_j, x_r) \end{aligned}$$

- We can thus compute the closest prototype to each input $\Phi(x_i)$, and update the partition matrix $\mathbf{U}$ accordingly.





## Time complexity

- In the right-hand side of the equation

$$\|\Phi(x_i) - v_k^{\Phi}\|^2 = \mathcal{K}(x_i, x_i) - 2 \sum_{j=1}^{n} \gamma_{jk} \mathcal{K}(x_i, x_j) + \sum_{j,r} \gamma_{jk} \gamma_{rk} \mathcal{K}(x_j, x_r)$$

  - The first term does not depend on $k$ and need not be computed (it is equal to 1 in the case of an RBF kernel).
  - The second terms costs $O(n)$ operations
  - The third term is fixed for cluster $k$ and can be calculated only once
- Consequently, each iteration of the kernel $c$-means algorithm requires $O(n^2)$ operations, which precludes its application to very large datasets.
- In contrast, each iteration of the standard $c$-means algorithm costs only $O(n)$ operations.

## Kernel $c$-means algorithm

1. Start with an initial partition $\mathbf{U}$ in $c$ clusters
2. Compute the distances between input vectors $x_i$ and prototypes $v_k^\Phi$ as

$$\|\Phi(x_i) - v_k^\Phi\|^2 = \mathcal{K}(x_i, x_i) - 2\sum_{j=1}^{n} \gamma_{jk}\mathcal{K}(x_i, x_j) + \sum_{j,r} \gamma_{jk}\gamma_{rk}\mathcal{K}(x_j, x_r)$$

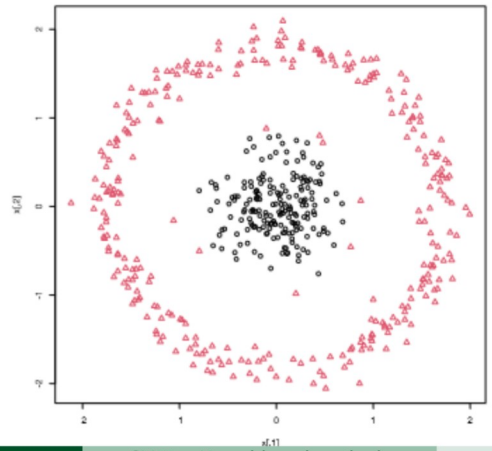
   with $\gamma_{ik} = u_{ik} / \sum_{j=1}^{n} u_{jk}$
3. Assign each $x_i$ to the cluster with the nearest prototype and update $\mathbf{U}$
4. Return to Step 3 while the partition has changed.





## Example in R

```
library(kernlab)
clus<-kkmeans(x,2,kernel="rbfdot")
plot(x,col=clus,pch=clus)
```





## Overview

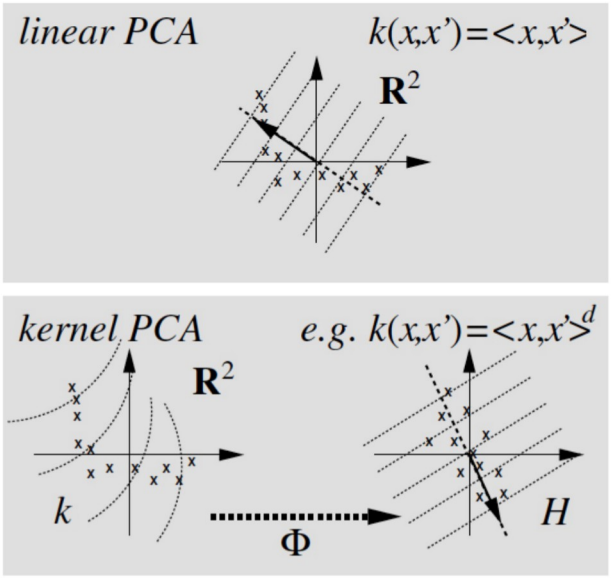




## Kernel-based feature extraction

- The present section describes a kernel-based method for performing a nonlinear form of Principal Component Analysis, called Kernel PCA.
- Using the kernel trick, we can efficiently compute principal components in a high-dimensional feature space, which is 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 features from possibly high-dimensional data sets. It is readily performed by solving an eigenvalue problem.
- PCA perform a rotation 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,

$$\mathbf{S} = \frac{1}{n} \sum_{i=1}^{n} x_i x_i^T \tag{10}$$

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

## Diagonalizing S

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

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

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

- Substituting (10) into this expression,

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

- 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 $\lambda u = \mathbf{S}u$ we may consider the $n$ projected equations

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





## 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.
- As before, map $\Phi$ will be defined implicitly by a kernel function $\mathcal{K}$.





## 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

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

(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
$$\lambda u = \mathbf{S}u$$
- 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
$$\lambda \langle \Phi(x_k), u \rangle = \langle \Phi(x_k), \mathbf{S}u \rangle, \quad k = 1, \ldots, n \tag{11}$$
  2. There exist coefficients $\alpha_i$, $i = 1, \ldots, n$ such that
$$u = \sum_{i=1}^{n} \alpha_i \Phi(x_i) \tag{12}$$





## Dual eigenvector representation

Using the bilinearity of the dot product and combining (11) and (12), we get

$$\lambda \sum_{i=1}^{n} \alpha_i \langle \Phi(x_k), \Phi(x_i) \rangle = \left\langle \Phi(x_k), \sum_{i=1}^{n} \alpha_i \mathbf{S} \Phi(x_i) \right\rangle \tag{13a}$$

$$= \frac{1}{n} \sum_{i=1}^{n} \alpha_i \left\langle \Phi(x_k), \sum_{j=1}^{n} \Phi(x_j) \langle \Phi(x_j), \Phi(x_i) \rangle \right\rangle \tag{13b}$$

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





## Dual eigenvector representation (continued)

- Let $\mathbf{K}$ be the $n \times n$ Gram matrix with general term $K_{ij} = \langle \Phi(x_i), \Phi(x_j) \rangle = \mathcal{K}(x_i, x_j)$.
- Eqs (13) can be written as
$$n\lambda \mathbf{K}\alpha = \mathbf{K}^2 \alpha \tag{14}$$
- Proof:
$$(\mathbf{K}\alpha)_k = \sum_i K_{ki} \alpha_i = \sum_i \alpha_i \langle \Phi(x_k), \Phi(x_i) \rangle$$
$$(\mathbf{K}^2\alpha)_k = \sum_i (\mathbf{K}^2)_{ki} \alpha_i = \sum_i \alpha_i \sum_j K_{kj} K_{ji}$$
$$= \sum_i \alpha_i \sum_j \langle \Phi(x_k), \Phi(x_j) \rangle \langle \Phi(x_j), \Phi(x_i) \rangle$$
$$= \sum_{i=1}^{n} \alpha_i \left\langle \Phi(x_k), \sum_j \Phi(x_j) \langle \Phi(x_j), \Phi(x_i) \rangle \right\rangle$$





## Dual eigenvector representation (continued)

- To find the solutions of (14) we solve the eigenvalue problem
$$n\lambda\alpha = \mathbf{K}\alpha \tag{15}$$
- Let $\lambda_1 \geq \ldots \geq \lambda_n$ denote the eigenvalues of $\mathbf{K}$ (the values $n\lambda$ in (15)), and $\alpha^1, \ldots, \alpha^n$ the corresponding eigenvectors.
- Let $\lambda_q$ 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 (12) and (15), 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 $\alpha^m$ by

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





## Feature extraction

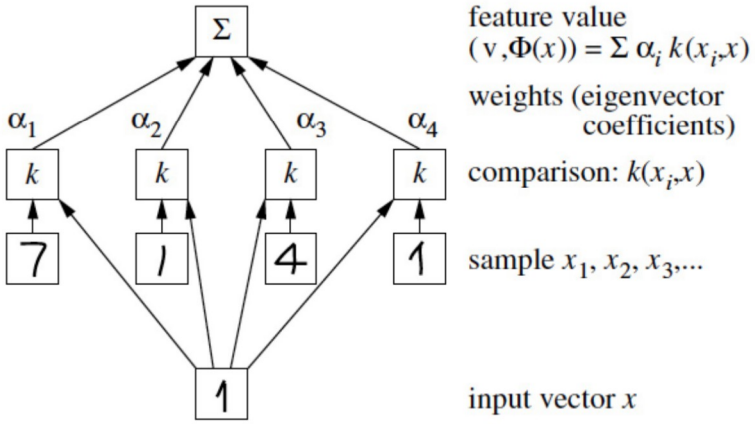
- For feature extraction, we need to compute the projections onto the eigenvectors $u^m$ in $\mathcal{H}$.
- Let $x$ be a test vector. The $m$-th feature is

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





## Feature extractor constructed using Kernel PCA




In the first layer, the input vector is compared to training vectors via a kernel function. The outputs are then linearly combined using weights $\alpha_i^m$.





## Summary of KPCA

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

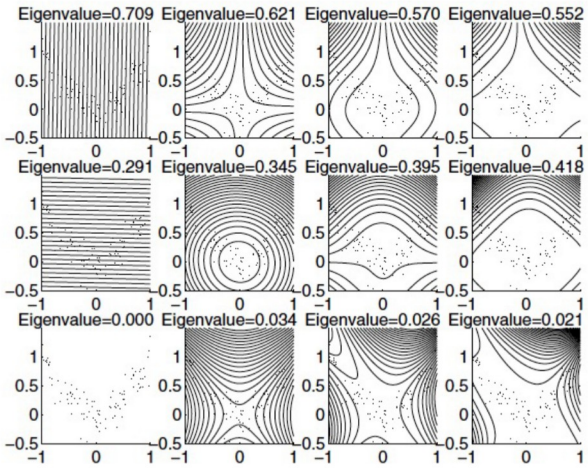
$$\widetilde{\mathcal{K}}(x_i, x) = \mathcal{K}(x_i, x) - \frac{1}{n}\sum_{k=1}^{n}\mathcal{K}(x_k, x) - \frac{1}{n}\sum_{k=1}^{n}\mathcal{K}(x_i, x_k) + \frac{1}{n^2}\sum_{k,\ell}^{n}\mathcal{K}(x_k, x_\ell)$$





## Overview

## Two-dimensional toy example with polynomial kernel



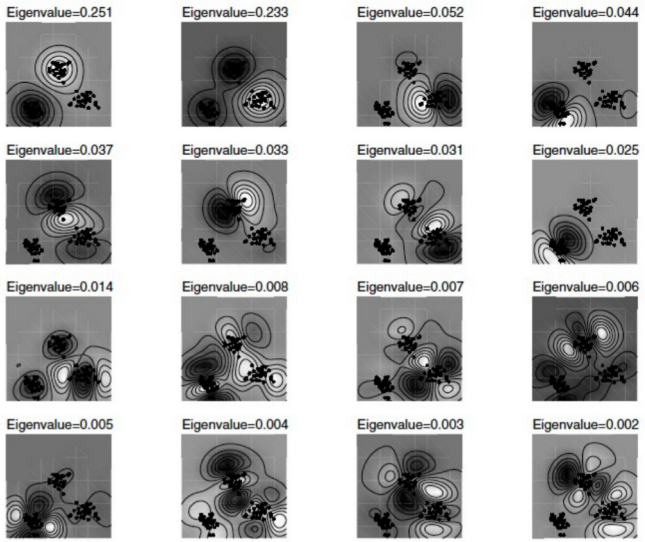

From left to right, the polynomial degree in the kernel increases from 1 to 4; from top to bottom, the first 3 eigenvectors are shown, in order of decreasing eigenvalue size (eigenvalues are normalized to sum to 1).





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

First 16 nonlinear principal components

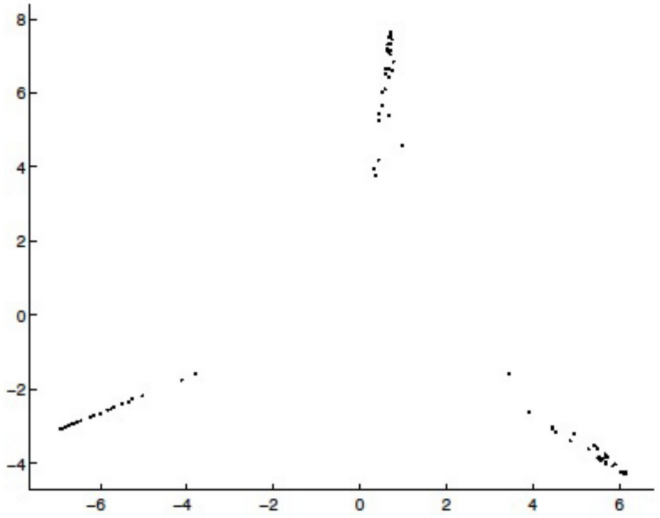







## Example with 3 clusters and radial kernel ($\gamma = 0.1$)

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 (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).





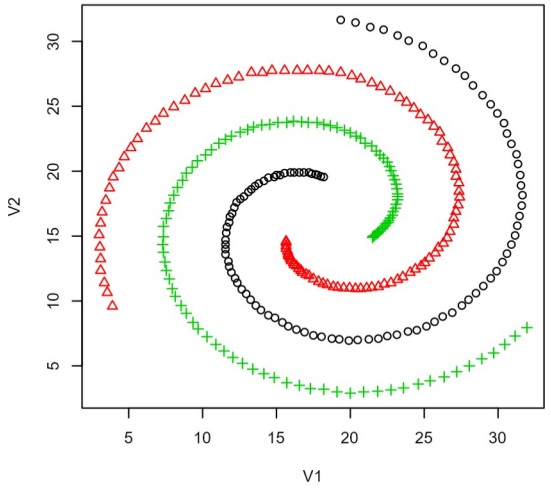
## KPCA in R

```
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")
```





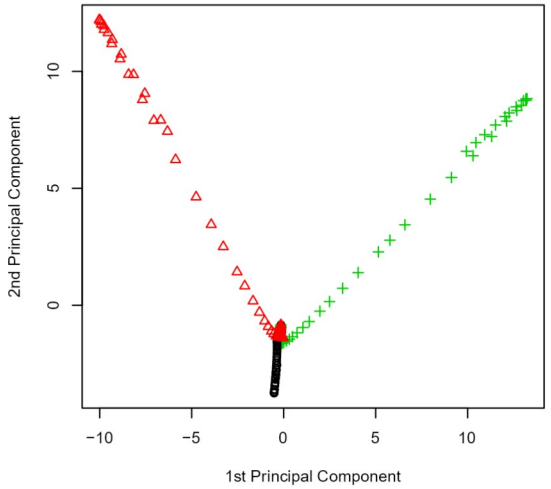
## Spirals data



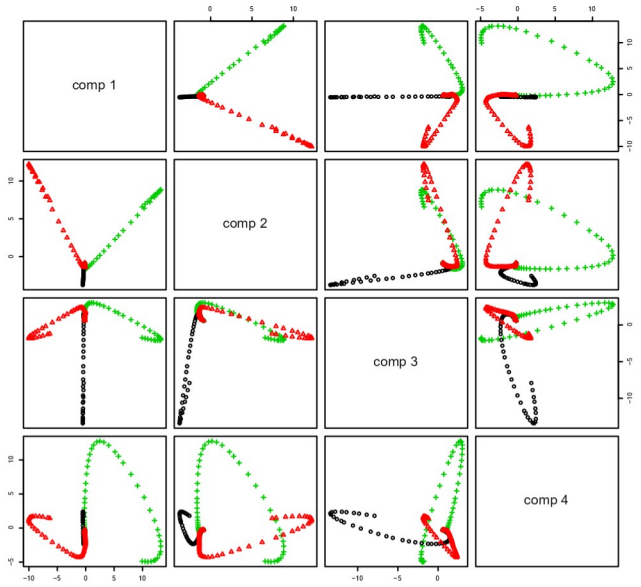





## First two principal components



# First four principal components








# Cumulated variance