Kernel-Based Learning & Multivariate Modeling

MIRI Master

Lluís A. Belanche
belanche@cs.upc.edu

Soft Computing Research Group

Universitat Politècnica de Catalunya

2019-2020

Kernel-Based Learning & Multivariate Modeling

Syllabus

Sep 10 Introduction to kernel-based learning
Sep 17 The SVM for classification, regression & novelty detection (I)
Oct 01 The SVM for classification, regression & novelty detection (II)
Oct 08 Kernel design (I): theoretical issues
Oct 15 Kernel design (II): practical issues
Oct 22 Kernelizing ML & stats algorithms
Oct 29 Advanced topics

Kernelizing ML & stats algorithms

[Q] What is "kernelizing"?

[A] do the inner product/distance in feature space:

- If an algorithm is **inner product-based**, the idea is:
  1. substitute $\left\langle \boldsymbol{x}_i, \boldsymbol{x}_j \right\rangle$ by $\left\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \right\rangle_{\mathcal{H}}$
  2. replace this by $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$

- If an algorithm is (Euclidean) **distance-based**, the idea is:
  1. substitute $||\boldsymbol{x}_i - \boldsymbol{x}_j||$ by $||\phi(\boldsymbol{x}_i) - \phi(\boldsymbol{x}_j)||_{\mathcal{H}}$
  2. replace this by $\sqrt{k(\boldsymbol{x}_i, \boldsymbol{x}_i) + k(\boldsymbol{x}_j, \boldsymbol{x}_j) - 2k(\boldsymbol{x}_i, \boldsymbol{x}_j)}$

Kernelizing ML & stats algorithms

A problem with the inner product

- If two data points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are translated as:

$$\begin{cases} \boldsymbol{x}_i \leftarrow \boldsymbol{x}_i - \boldsymbol{x}_0 \\ \boldsymbol{x}_j \leftarrow \boldsymbol{x}_j - \boldsymbol{x}_0 \end{cases}$$

  then the inner product $\left\langle \boldsymbol{x}_i, \boldsymbol{x}_j \right\rangle$ changes substantially

- This is not suitable for algorithms that should be **translation-invariant** (e.g. PCA), unless we center the data in feature space

Kernelizing ML & stats algorithms

Conditional positive semi-definiteness

A symmetric function k is called **conditionally positive semi-definite** (CPSD) in $\mathcal{X}$ if for every $n \in \mathbb{N}$, and every choice $x_1, \ldots, x_n \in \mathcal{X}$, the matrix $\mathbf{K} = (k_{ij})$, where $k_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is CPSD.

A real symmetric matrix $A_{n \times n}$ is CPSD if and only if $\forall \boldsymbol{c} \in \mathbb{R}^N$ such that $\boldsymbol{c}^\mathsf{T} \mathbf{1} = 0,\ \boldsymbol{c}^\mathsf{T} A \boldsymbol{c} \geq 0$.

Example: in $\mathcal{X} = \mathbb{R}^d$, $k(\boldsymbol{x}, \boldsymbol{x'}) = -\|\boldsymbol{x} - \boldsymbol{x'}\|^2$ is CPSD (but not PSD)

Kernelizing ML & stats algorithms

The kernel trick for distances

Theorem. Let k be a CPSD kernel on $\mathcal{X}$, satisfying $k(\boldsymbol{x}, \boldsymbol{x}) = 0$ for all $\boldsymbol{x} \in \mathcal{X}$. Then there exists a Hilbert space $\mathcal{H}$ and a mapping $\phi : \mathcal{X} \to \mathcal{H}$ such that

$$k(\boldsymbol{x}, \boldsymbol{x'}) = -\|\phi(\boldsymbol{x}) - \phi(\boldsymbol{x'})\|^2$$

It turns out that it suffices for a kernel to be CPSD! Since the class of CPSD kernels is larger than that of PSD kernels, a larger set of learning algorithms are prone to kernelization.

Kernelizing ML & stats algorithms

Example: k-nearest neighbours

(kNN classifies new examples by finding the k closest examples in the sample and taking a majority vote)

$$
\begin{aligned}
k(\boldsymbol{x},\boldsymbol{x}) + k(\boldsymbol{x'},\boldsymbol{x'}) - 2k(\boldsymbol{x},\boldsymbol{x'}) = \\
\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}) \rangle + \left\langle \phi(\boldsymbol{x'}), \phi(\boldsymbol{x'}) \right\rangle - 2\left\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x'}) \right\rangle = \\
\|\phi(\boldsymbol{x})\|^2 + \|\phi(\boldsymbol{x'})\|^2 - 2\left\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x'}) \right\rangle = \\
\|\phi(\boldsymbol{x}) - \phi(\boldsymbol{x'})\|^2 =: d^2_{\mathcal{H}}(\boldsymbol{x},\boldsymbol{x'})
\end{aligned}
$$

- This (square) Euclidean distance in *feature space* can be calculated using 3 calls to the kernel function (or 1 if k is normalized), for k PSD

- Note that $\sqrt{-k(\boldsymbol{x},\boldsymbol{x'})}$ is also a Euclidean distance, for k CPSD

Kernelizing ML & stats algorithms

Standard PCA

- We are given a data set $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, $\boldsymbol{x}_i \in \mathbb{R}^d$ for $i = 1, \ldots n$ which is centered around the origin, *i.e.* $\sum_{i=1}^{n} \boldsymbol{x}_i = 0$

- The sample covariance matrix C of the data is defined as

$$C = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{x}_i^{\mathsf{T}}$$

Kernelizing ML & stats algorithms

Standard PCA

What are the PCs?

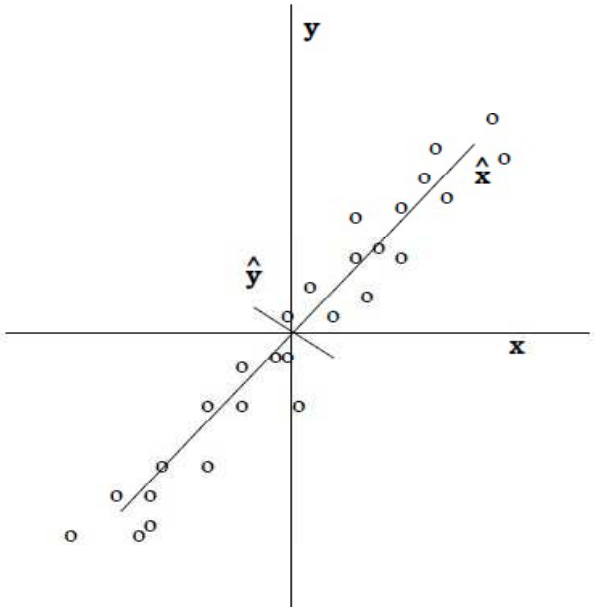
1. The goal of PCA is to replace the original axes with the principal components (PC) of the data

2. We choose the first PC as a **projection direction** such that the projections of the data onto it have **maximum variance**

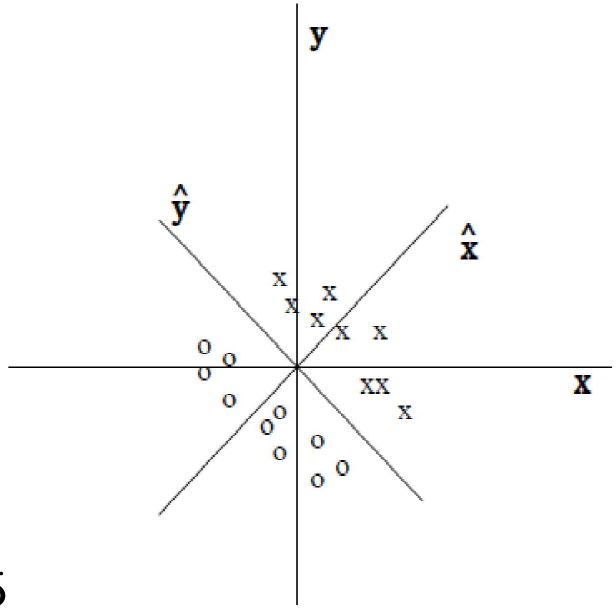
3. All subsequent components can be defined as **orthogonal** projection directions with the next largest variance

Kernelizing ML & stats algorithms

Standard PCA



482424434



295393335

Left: when data lies mainly in some lower-dimensionnal hyperplane of $\mathbb{R}^l$ then we can use only the first $l < d$ PCs of our data to lower the dimension from d to l without losing too much information –in fact we *may* only loose noise (here $d = 2, l = 1$)

Right: when data are not clearly separable in any particular axis of our original space we may redefine the axes such that they are. In the plot, data is not separable on either x or y, but it is separable on $\widehat{x}$

Kernelizing ML & stats algorithms

Standard PCA

- The direction $\boldsymbol{v}$ of maximum variance is the (unique) solution to the optimization problem:

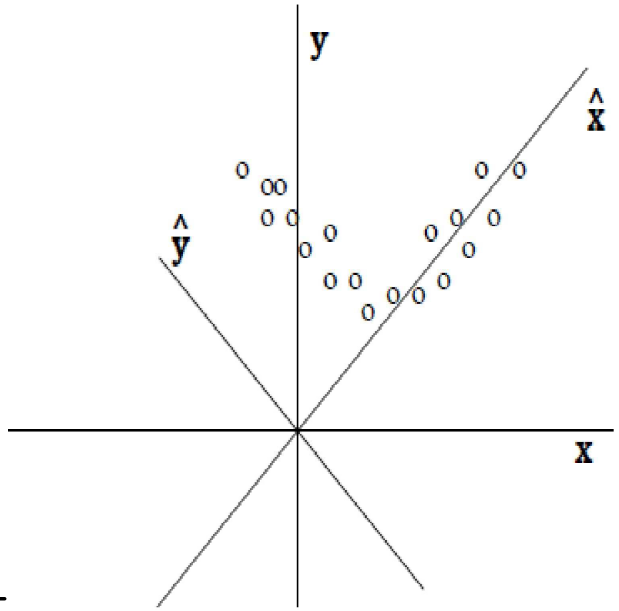
$$\arg\max_{\boldsymbol{v}^\mathsf{T}\boldsymbol{v}=1} \left\{ \boldsymbol{v}^\mathsf{T} C \boldsymbol{v} \right\} = \arg\max_{\boldsymbol{v}} \left\{ \frac{\boldsymbol{v}^\mathsf{T} C \boldsymbol{v}}{\boldsymbol{v}^\mathsf{T}\boldsymbol{v}} \right\}$$

- The solution is given by the eigenvector $\boldsymbol{v}$ corresponding to the largest eigenvalue of C: $C\boldsymbol{v} = \lambda_{(1)}\boldsymbol{v}$

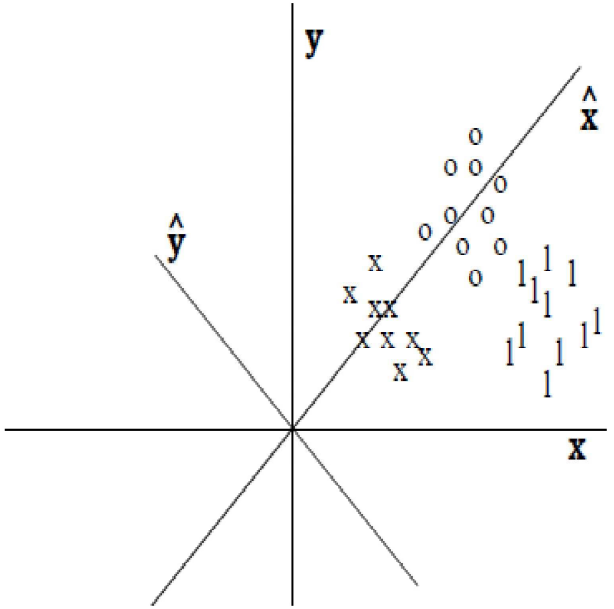
- Subsequent PCs correspond to the d (orthogonal) eigenvectors of C ordered with decreasing eigenvalue $\lambda_{(1)} > \lambda_{(2)} \ldots > \lambda_{(d)} \geq 0$

Kernelizing ML & stats algorithms

Kernel PCA from Standard PCA



482424334



295393335

Left: What if the data does not fall neatly into some hyperplane?

Right: What if our data falls into more than d clusters (we do not have enough PCs to make all the data separable)?

Kernelizing ML & stats algorithms

Kernel PCA from Standard PCA

- If we could define our principal components to be arbitrary **manifolds** we could get higher variance and better separability

- If we first (non-linearly) map our data onto a higher-dimensional space where the data falls neatly onto some hyperplane we can perform Standard PCA in that space

- These PCs will map back onto the arbitrary manifolds that we need in our lower-dimensional space

Kernelizing ML & stats algorithms

Kernel PCA from Standard PCA

Kernel PCA allows us to perform PCA in this higher dimension using the kernel trick, doing all our calculations in a lower dimension.

Recall the idea of mapping input data into some Hilbert space (called the *feature space*) via a non-linear mapping $\phi : \mathcal{X} \to \mathcal{H}$:

$\to$ the new sample covariance matrix C of the data is given by

$$C = \frac{1}{n} \sum_{i=1}^{n} \phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i)^{\mathsf{T}}$$

Kernelizing ML & stats algorithms

Kernel PCA from Standard PCA

To find the first PC, again we need to solve $C\boldsymbol{v} = \lambda\boldsymbol{v}$. Therefore:

$$\frac{1}{n}\sum_{i=1}^{n}\phi(\boldsymbol{x}_i)\left(\phi(\boldsymbol{x}_i)^\top\boldsymbol{v}\right) = \lambda\boldsymbol{v}$$

Since $\lambda \neq 0$, $\boldsymbol{v}$ must be in the span of the set of vectors $\phi(\boldsymbol{x}_i)$, *i.e.* it can be written as some linear combination thereof:

$\rightarrow$ there must exist some α_i such that $\boldsymbol{v} = \sum_{i=1}^{n}\alpha_i\phi(\boldsymbol{x}_i)$.

Kernelizing ML & stats algorithms

Kernel PCA from Standard PCA

$$\lambda \sum_{j=1}^{n} \alpha_j \phi(\boldsymbol{x}_j) = \frac{1}{n} \sum_{i=1}^{n} \phi(\boldsymbol{x}_i) \phi(\boldsymbol{x}_i)^\top \sum_{j=1}^{n} \alpha_j \phi(\boldsymbol{x}_j)$$

switch to our alternative notation ...

$$\lambda \sum_{j=1}^{n} \alpha_j \phi(\boldsymbol{x}_j) = \frac{1}{n} \sum_{i=1}^{n} \phi(\boldsymbol{x}_i) \left\langle \phi(\boldsymbol{x}_i), \sum_{j=1}^{n} \alpha_j \phi(\boldsymbol{x}_j) \right\rangle$$

Kernelizing ML & stats algorithms

Kernel PCA from Standard PCA

and the trick ... introduce inner product with $\phi(\boldsymbol{x}_k)$ for an arbitrary k on both sides ($1 \leq k \leq n$):

$$\left\langle \lambda \sum_{j=1}^{n} \alpha_j \phi(\boldsymbol{x}_j), \phi(\boldsymbol{x}_k) \right\rangle = \lambda \sum_{j=1}^{n} \alpha_j \left\langle \phi(\boldsymbol{x}_j), \phi(\boldsymbol{x}_k) \right\rangle$$

$$\left\langle \frac{1}{n} \sum_{i=1}^{n} \phi(\boldsymbol{x}_i) \left\langle \phi(\boldsymbol{x}_i), \sum_{j=1}^{n} \alpha_j \phi(\boldsymbol{x}_j) \right\rangle, \phi(\boldsymbol{x}_k) \right\rangle = \left\langle \frac{1}{n} \sum_{i=1}^{n} \phi(\boldsymbol{x}_i) \sum_{j=1}^{n} \alpha_j \left\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \right\rangle, \phi(\boldsymbol{x}_k) \right\rangle$$

$$= \left\langle \frac{1}{n} \sum_{j=1}^{n} \alpha_j \sum_{i=1}^{n} \phi(\boldsymbol{x}_i) \left\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \right\rangle, \phi(\boldsymbol{x}_k) \right\rangle = \frac{1}{n} \sum_{j=1}^{n} \alpha_j \sum_{i=1}^{n} \left\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_k) \right\rangle \left\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \right\rangle$$

Kernelizing ML & stats algorithms

Kernel PCA from Standard PCA

Now we can introduce the kernel; since

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \phi(\boldsymbol{x})^\top \phi(\boldsymbol{x'}) = \left\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x'}) \right\rangle_{\mathcal{H}}, \ \boldsymbol{x}, \boldsymbol{x'} \in \mathcal{X}$$

$$\lambda \sum_{j=1}^{n} \alpha_j \left\langle \phi(\boldsymbol{x}_j), \phi(\boldsymbol{x}_k) \right\rangle = \frac{1}{n} \sum_{j=1}^{n} \alpha_j \sum_{i=1}^{n} \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_k) \rangle \left\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \right\rangle$$

is rewritten as:

$$\lambda \sum_{j=1}^{n} \alpha_j k(\boldsymbol{x}_j, \boldsymbol{x}_k) = \frac{1}{n} \sum_{j=1}^{n} \alpha_j \sum_{i=1}^{n} k(\boldsymbol{x}_i, \boldsymbol{x}_k) k(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

Kernelizing ML & stats algorithms

Kernel PCA from Standard PCA

Now let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)^\mathsf{T}$ and $\mathbf{K} = (k_{ij})$, where $k_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$:

$$\lambda \mathbf{K} \boldsymbol{\alpha} = \frac{1}{n} \mathbf{K}^2 \boldsymbol{\alpha}$$

This is again an eigenvalue problem! To see this, put $\boldsymbol{p} := \mathbf{K}\boldsymbol{\alpha}$ and rewrite:

$$\frac{1}{n} \mathbf{K} \boldsymbol{p} = \lambda \boldsymbol{p}$$

where $\boldsymbol{p}, \lambda$ are the eigenvectors and eigenvalues of $\frac{1}{n}\mathbf{K}$ (we get rid of $\boldsymbol{\alpha}$)

$\frac{1}{n}\mathbf{K}$ has the same eigenvectors/values as $\mathbf{K}$, with eigenvalues scaled by n

Kernelizing ML & stats algorithms

Kernel PCA from Standard PCA

- In Standard PCA, the matrix XX^{T} grows with the **dimension** of the data points d

- In Kernel PCA, the matrix $\mathbf{K}$ grows with the **number** of data points n; in consequence, we get n non-linear PCs

- When the kernel function is standard dot product, Kernel PCA solution reduces to Standard PCA

Kernelizing ML & stats algorithms

Kernel PCA from Standard PCA

- Centered data is required to perform an effective PCA
- Even though the $\{\boldsymbol{x}_i\}$ were centered, the $\{\phi(\boldsymbol{x}_i)\}$ are *not* guaranteed to be centered in feature space!
- We have to "center" the kernel matrix before doing Kernel PCA:

$$\mathbf{K} := \mathbf{K} - \frac{1}{n}\mathbf{1}\mathbf{K} - \frac{1}{n}\mathbf{K}\mathbf{1} + \frac{1}{n^2}\mathbf{1}\mathbf{K}\mathbf{1}$$

where $\mathbf{1}$ is a $n \times n$ matrix of ones

Kernelizing ML & stats algorithms

An illustration of KPCA

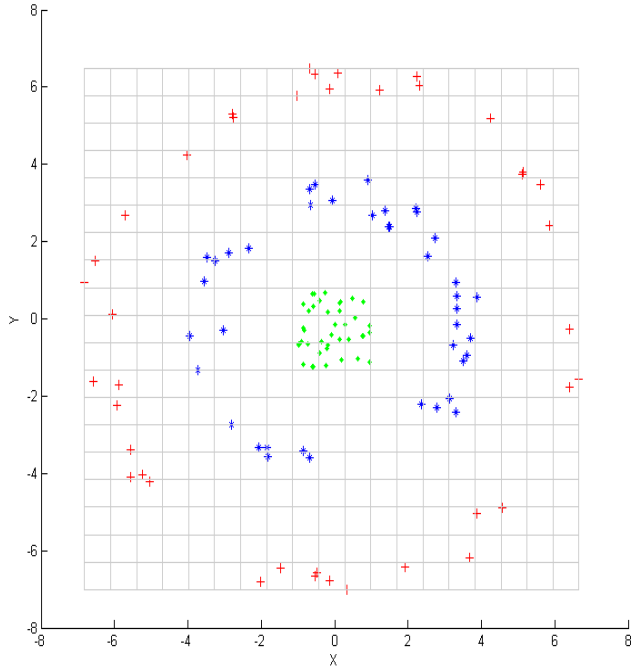
Left (input space): Green lines are desired non-linear projections

Right (feature space): Green lines are linear projections onto the first non-linear PC

(from *Pattern Recognition and Machine Learning*, C. M. Bishop, Springer, 2006)

Kernelizing ML & stats algorithms

Example



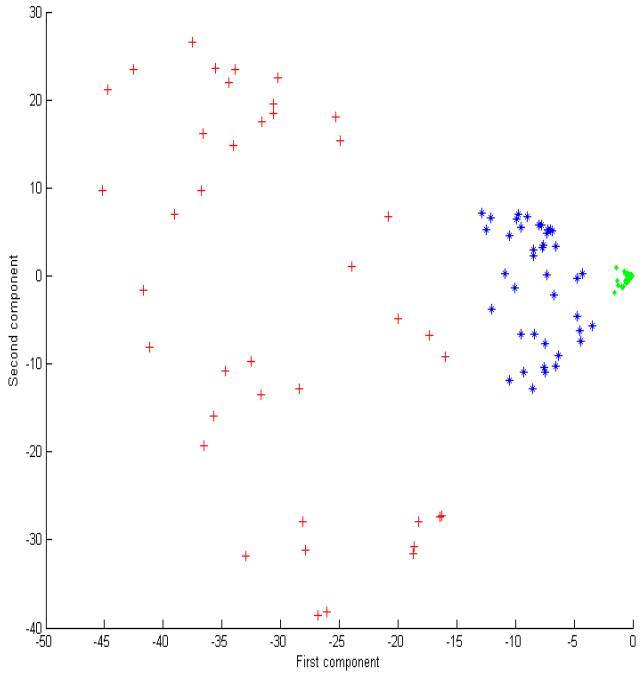
5430568466

Three concentric clouds of points

(Source: Wikipedia, Petter Strandmark)

Kernelizing ML & stats algorithms

Example



5125580446

$$k(\boldsymbol{x}, \boldsymbol{x'}) = (\left\langle \boldsymbol{x}, \boldsymbol{x'} \right\rangle + 1)^2$$

(note that the color of the points is not part of the algorithm)

(Source: Wikipedia, Petter Strandmark)

Kernelizing ML & stats algorithms

Bad news

In Standard PCA, it is common to:

1. use the eigenvalues to rank the eigenvectors based on how much of the data variation is captured by each PC

2. estimate a "convenient" number of PCs and perform dimensionality reduction by projecting the original data onto the reduced set of eigenvectors

We cannot compute the kPCs (since they reside in the high-dimensional feature space), only the projections of our data onto the kPCs; however, we can perform new projections on test data (as in Standard PCA)

Kernelizing ML & stats algorithms

Summary of Standard PCA

1. We are given a data set of d-dimensional vectors $X = \{\boldsymbol{x}_i\}$ for $i = 1, \ldots, n$ which we **center** around the origin, as $\boldsymbol{x}_i := \boldsymbol{x}_i - \frac{1}{n}\sum_{j=1}^{n} \boldsymbol{x}_j$

2. Compute the **sample covariance matrix** of the data as $C = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{x}_i^{\mathsf{T}}$

3. Compute the **eigenvalues** and **eigenvectors** $(\lambda_i, \boldsymbol{v}_i)$ of C; **decide** a number $l < d$

4. **Let** $\widehat{\boldsymbol{x}}_i = V\boldsymbol{x}_i$, where $V = [\boldsymbol{v}_1; \ldots; \boldsymbol{v}_l]$

5. **Return** the l-dimensional data sample $\widehat{X} = \{\widehat{\boldsymbol{x}}_i\}$ for $i = 1, \ldots, n$

Kernelizing ML & stats algorithms

Summary of Kernel PCA

1. We are given a data set of objects $X = \{\boldsymbol{x}_i\}$ for $i = 1, \ldots, n$

2. Choose a **kernel** function k and compute the **kernel matrix** $\mathbf{K}$

3. **Center** the kernel matrix as: $\mathbf{K} := \mathbf{K} - \frac{1}{n}\mathbf{1}\mathbf{K} - \frac{1}{n}\mathbf{K}\mathbf{1} + \frac{1}{n^2}\mathbf{1}\mathbf{K}\mathbf{1}$

4. Compute the **eigenvalues** and **eigenvectors** $(\lambda_i, \boldsymbol{v}_i)$ of $\mathbf{K}$; **decide** a number $l < n$. Set $\boldsymbol{\alpha}_j = \boldsymbol{v}_j / \sqrt{\lambda_j}$ and let $\widehat{\boldsymbol{x}}_i = \left(\sum_{k=1}^{n} \boldsymbol{\alpha}_{jk}\, k(\boldsymbol{x}_i, \boldsymbol{x}_k) \right)_{j=1}^{l}$

5. **Return** the l-dimensional data sample $\widehat{X} = \{\widehat{\boldsymbol{x}}_i\}$ for $i = 1, ..., n$

Kernelizing ML & stats algorithms

Key aspects of kernel methods

Kernel-based methods consist of two ingredients:

1. The kernel function (this is non-trivial)

2. The algorithm taking kernels as input
   - Data items are embedded into a vector space (feature space FS)
   - Linear relations are sought among the elements of the FS
   - The coordinates of these images are not needed (and are usually unknown): only their pairwise inner products matter
   - These inner products can sometimes be computed efficiently and implicitly in the input space (kernel function)
   - This requires expressing a problem wherein the data appear in the form of inner products or distances
   - The solution vector is expressed as a linear combination of kernel evaluations centered at the data

Kernelizing ML & stats algorithms

Work so far ...

Many (classical and new) learning algorithms can be "kernelized":

- Support Vector Machine (SVM) and Relevance Vector Machine (SVM)
- Fisher Discriminant Analysis (kFDA), Principal Components Analysis (kPCA) and Canonical Correlation Analysis (kCCA), Independent Component Analysis (kICA)
- Kernel (regularized) linear and logistic regression
- kernel kNN (k Nearest Neighbours)
- Clustering (Spectral Clustering, Kernel k-means)
- PLS, Parzen Windows, Vector Quantization, ...
- Statistical indexes (e.g. Kendall's tau correlation coefficient to measure similarity between permutations)