Kernel principal component analysis

From Wikipedia, the free encyclopedia

In the field of multivariate statistics, **kernel principal component analysis (kernel PCA)** [1] is an extension of principal component analysis (PCA) using techniques of kernel methods. Using a kernel, the originally linear operations of PCA are done in a reproducing kernel Hilbert space with a non-linear mapping.

Contents

- 1 Background: Linear PCA
- 2 Introduction of the Kernel to PCA
- 3 Large Datasets
- 4 Example
- 5 Applications
- 6 See also
- 7 References

Background: Linear PCA

Recall that conventional PCA operates on zero-centered data; that is,

$$\frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i = \mathbf{0}.$$

It operates by diagonalizing the covariance matrix,

$$C = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i\mathbf{x}_i^\top$$

in other words, it gives an eigendecomposition of the covariance matrix:

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

which can be rewritten as

$$\lambda \mathbf{x}_i^\top \mathbf{v} = \mathbf{x}_i^\top C\mathbf{v} \quad \forall i \in [1, N].$$[2]

(See also: Covariance matrix as a linear operator)

Introduction of the Kernel to PCA

To understand the utility of kernel PCA, particularly for clustering, observe that, while *N* points cannot in general be linearly separated in $d < N$ dimensions, they can almost always be linearly separated in $d \geq N$ dimensions. That is, given *N* points, $\mathbf{x}_i$, if we map them to an *N*-dimensional space with

$\Phi(\mathbf{x}_i)$ where $\Phi : \mathbb{R}^d \to \mathbb{R}^N$,

it is easy to construct a hyperplane that divides the points into arbitrary clusters. Of course, this Φ creates linearly independent vectors, so there is no covariance on which to perform eigendecomposition *explicitly* as we would in linear PCA.

Instead, in kernel PCA, a non-trivial, arbitrary Φ function is 'chosen' that is never calculated explicitly, allowing the possibility to use very-high-dimensional Φ's if we never have to actually evaluate the data in that space. Since we generally try to avoid working in the Φ-space, which we will call the 'feature space', we can create the N-by-N kernel

$$K = k(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x}), \Phi(\mathbf{y})) = \Phi(\mathbf{x})^T \Phi(\mathbf{y})$$

which represents the inner product space (see Gramian matrix) of the otherwise intractable feature space. The dual form that arises in the creation of a kernel allows us to mathematically formulate a version of PCA in which we never actually solve the eigenvectors and eigenvalues of the covariance matrix in the $\Phi(\mathbf{x})$-space (see Kernel trick). The N-elements in each column of *K* represent the dot product of one point of the transformed data with respect to all the transformed points (N points). Some well-known kernels are shown in the example below.

Because we are never working directly in the feature space, the kernel-formulation of PCA is restricted in that it computes not the principal components themselves, but the projections of our data onto those components. To evaluate the projection from a point in the feature space $\Phi(\mathbf{x})$ onto the kth principal component V (where superscript k means the component k, not powers of k)

$$\mathbf{V}^{k^T} \Phi(\mathbf{x}) = \left(\sum_{i=1}^{N} \mathbf{a_i}^k \Phi(\mathbf{x_i}) \right)^T \Phi(\mathbf{x})$$

We note that $\Phi(\mathbf{x_i})^T \Phi(\mathbf{x})$ denotes dot product, which is simply the elements of the kernel K. It seems all that's left is to calculate and normalize the $\mathbf{a_i}^k$, which can be done by solving the eigenvector equation

$$N\lambda \mathbf{a} = K\mathbf{a}$$

where N is the number of data points in the set, and λ and $\mathbf{a}$ are the eigenvalues and eigenvectors of K. Then to normalize the eigenvectors $\mathbf{a}^k$'s, we require that

$$1 = (\mathbf{a}^k)^T \mathbf{a}^k$$

Care must be taken regarding the fact that, whether or not x has zero-mean in its original space, it is not guaranteed to be centered in the feature space (which we never compute explicitly). Since centered data is required to perform an effective principal component analysis, we 'centralize' K to become K'

$$K' = K - \mathbf{1_N} K - K \mathbf{1_N} + \mathbf{1_N} K \mathbf{1_N}$$

where $\mathbf{1_N}$ denotes a N-by-N matrix for which each element takes value $1/N$. We use K' to perform the kernel PCA algorithm described above.

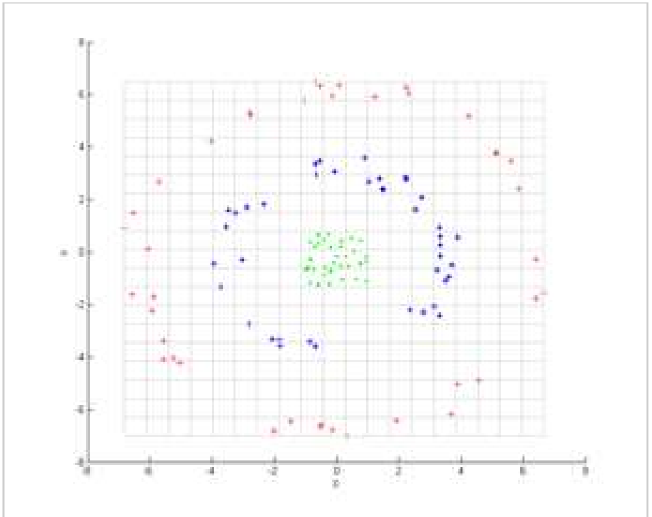
One caveat of kernel PCA should be illustrated here. In linear PCA, we can use the eigenvalues to rank the eigenvectors based on how much of the variation of the data is captured by each principal component. This is useful for data dimensionality reduction and it could also be applied to KPCA. However, in practice there are cases that all variations of the data are same. This is typically caused by a wrong choice of kernel scale.

Large Datasets

In practice, a large data set leads to a large K, and storing K may become a problem. One way to deal with this is to perform clustering on the dataset, and populate the kernel with the means of those clusters. Since even this method may yield a relatively large K, it is common to compute only the top P eigenvalues and eigenvectors of K.

Example

Consider three concentric clouds of points (shown); we wish to use kernel PCA to identify these groups. The color of the points is not part of the algorithm, but only there to show how the data groups together before and after the transformation.


Input points before kernel PCA

First, consider the kernel

$$k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^{\mathrm{T}} \boldsymbol{y} + 1)^2$$

Applying this to kernel PCA yields the next image.

Now consider a Gaussian kernel:

$$k(\boldsymbol{x}, \boldsymbol{y}) = e^{\frac{-||\boldsymbol{x}-\boldsymbol{y}||^2}{2\sigma^2}},$$

That is, this kernel is a measure of closeness, equal to 1 when the points coincide and equal to 0 at infinity.

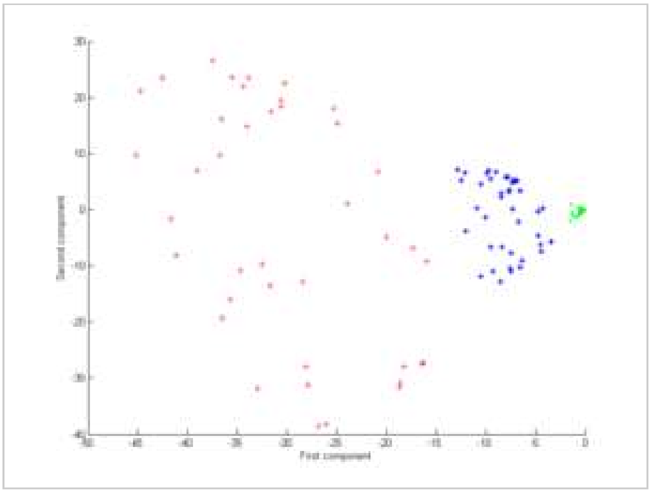
Note in particular that the first principal component is enough to distinguish the three different groups, which is impossible using only linear PCA, because linear PCA operates only in the given (in this case two-dimensional) space, in which these concentric point clouds are not linearly separable.

Applications

Kernel PCA has been demonstrated to be useful for novelty detection,[3] and image de-noising.[4]

See also

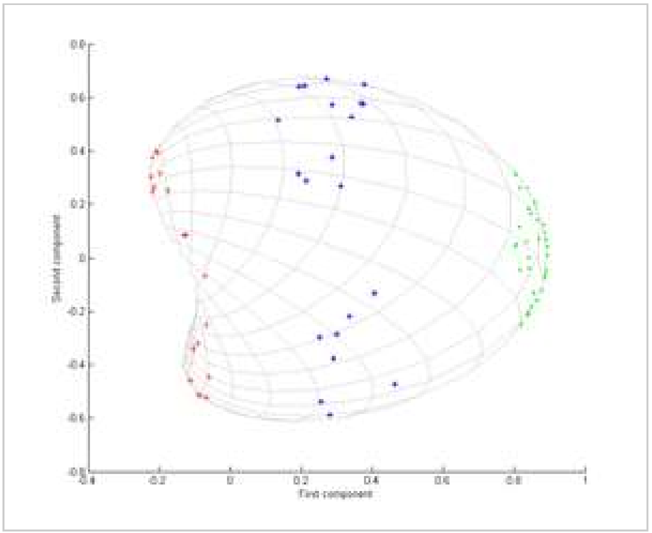
- Cluster analysis
- Kernel trick
- Multilinear PCA
- Multilinear subspace learning
- Nonlinear dimensionality reduction
- Spectral clustering



Output after kernel PCA with $k(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}+1)^2$. The three groups are distinguishable using the first component only.

References

1. Nonlinear Component Analysis as a Kernel Eigenvalue Problem (http://dx.doi.org/10.1162/089976698300017467)
2. Nonlinear Component Analysis as a Kernel Eigenvalue Problem (Technical Report) (http://www.face-rec.org/algorithms/Kernel/kernelPCA_scholkopf.pdf)
3. Kernel PCA for Novelty Detection. Pattern Recognition, 40, 863-874, 2007 (http://www.heikohoffmann.de/kpca.html)
4. Kernel PCA and De-Noising in Feature Spaces. NIPS, 1999 (http://citeseer.ist.psu.edu/old/mika99kernel.html)



Output after kernel PCA, with a Gaussian kernel.

Retrieved from "https://en.wikipedia.org/w/index.php?title=Kernel_principal_component_analysis&oldid=673787821"

Categories: Multivariate statistics | Signal processing | Machine learning algorithms | Kernel methods for machine learning