Kernel PCA

Schölkopf, Smola and Müller: Nonlinear Component Analysis as a Kernel Eigenvalue Problem

Ian Fasel

September 25, 2001

1 Overview

In this talk, I will discuss the kernel PCA paper by Schölkopf, Smola and Müller. I begin by providing some background on kernel methods and PCA, in order to motivate the discussion and give some intuition about what PCA is for and why we would want to use a kernel method with it. Then I go into some depth showing the derivation of kernel PCA, because I feel it is important to actually work through the equations at least once to see how the kernel trick is used and how it fits into standard PCA. Next, I will talk about some of the things that have to be done differently in kernel PCA vs. regular, linear PCA, which I hope will help illustrate some of the benefits of using a Mercer kernel as opposed to some other (possibly nonlinear) function. Next I go through the toy examples given by Schölkopf, Smola and Müller, in addition to the real-world character recognition task they present. Finally, I discuss the comparisons between kernel PCA and other techniques given in the paper.

Slide 1

Overview

- Backround:
  - High Dimensional Representations
  - Kernels
  - PCA
- Derivation of kernel PCA
- Differences between kernel PCA and regular PCA
- Examples:
  - Toy examples
  - Real World Example: Handwritten Character Recognition
- Comparisons with other techniques

2 High dimensionality can be good

I begin with an attempt to provide some intuition that taking raw input and transforming it into some higher dimensional space can be a good thing.

The framework we are using here is called Statistical learning theory, which I believe is synonymous with VC (Vapnik-Chervonenkis) theory. This theory attempts to provide a mathematical framework for answering questions about how to make decisions about e.g., classification given only example data, with no prior information or intuition about the problem.

One of the questions that is addressed in VC theory is: Under what conditions are high dimensional representations good? One of the best-known concepts of VC theory is *VC dimension*, which is in some sense a one-number summary of a learning machine's capacity. Others in this class will undoubtedly go into more depth later, however one result is that often mappings which take us into a higher dimensional space than the dimension of the input space provide us with greater classification power.

Slide 2

Under what conditions are high dimensional representations good?

Given some problem, how do we know what classes of functions are capable of solving that problem?

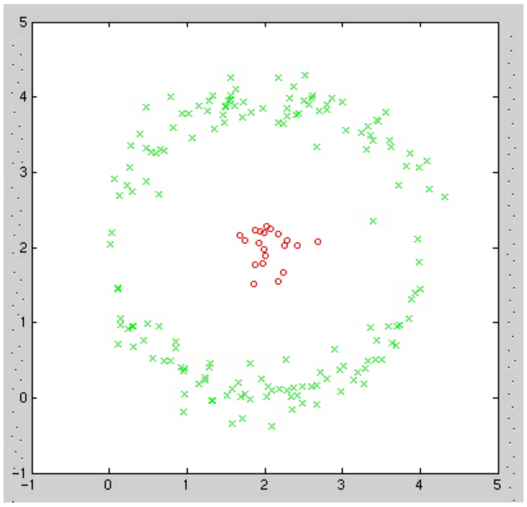
VC (Vapnik-Chervonenkis) theory tells us that often mappings which take us into a higher dimensional space than the dimension of the input space provide us with greater classification power.

2.1 Example: 2-D data: ring with center.

A nice example is as follows. Suppose we have the following data: (show Slide 3)

Slide 3

Example: Take a two class problem with data in $\mathbb{R}^2$.



These classes are linearly inseparable in the input space. Instead, they would have to be separated with a quadratic decision surface.

In this example, we perform a simple mapping from $\mathbb{R}^2$ to $\mathbb{R}^3$. Suppose we have the mapping:

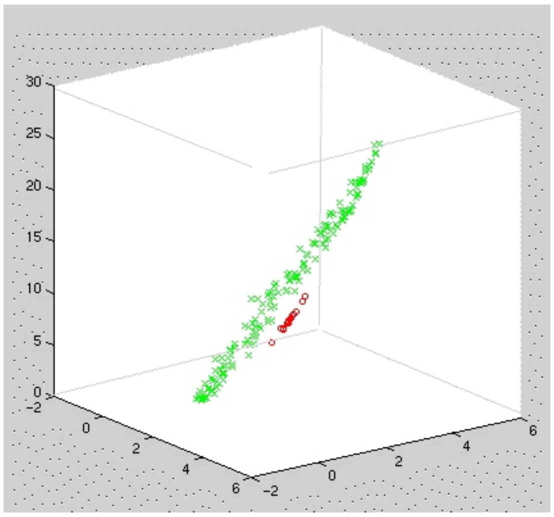
$$\begin{aligned} \Phi : \mathcal{X} = \mathbb{R}^2 &\to \mathcal{H} = \mathbb{R}^3 \\ (x_1, x_2) &\mapsto (x_1, x_2, x_1^2 + x_2^2) \end{aligned} \tag{1}$$

Then the ring gets spread out so that the center can be separated with a separating plane.

Slide 4

We can make the problem linearly separable by a simple mapping into $\mathbb{R}^3$

$$\begin{aligned} \Phi : \mathcal{X} = \mathbb{R}^2 &\to \mathcal{H} = \mathbb{R}^3 \\ (x_1, x_2) &\mapsto (x_1, x_2, x_1^2 + x_2^2) \end{aligned} \tag{2}$$



2.2 Problems with high-D

The problem of high dimensionality is that this can seriously increase computation time. Consider the product features used in the paper, which take all possible dth order products of the elements of the input vector and write them into a new vector. For instance, take the product feature extractor (from the paper) with $d = 2$ where $d = 2$,

This is fine for small toy examples, but for realistically sized problems: for N dimensional data, there exist different monomials comprising a feature space of dimension N_h. Thus, 16×16 pixel input images with a monomial degree $d = 5$ yeild a dimension of almost 10^{10}. This is bad.

Slide 5

So what's wrong with high-D?

High Dimensionality increases computational complexity. For instance, take the product feature extractor (from paper) with $d = 2$,

$$\begin{aligned} \Phi : \mathbb{R}^2 &\to \mathbb{R}^3 \\ (x_1, x_2) &\to (x_1^2, x_2^2, x_1x_2, x_2x_1). \end{aligned} \tag{3}$$

This is simple, however for N dimensional data, there exist

$$N_{\mathcal{H}} = \begin{pmatrix} N+d+1 \\ d \end{pmatrix} = \frac{(d+N-1)!}{d!(N-1)!} \tag{4}$$

different monomials comprising a feature space of dimension $N_{\mathcal{H}}$. Thus, 16×16 pixel input images with a monomial degree $d = 5$ yeild a dimension of almost 10^{10}. This is bad.!

Is there any way to get around this problem and still get the benefit of the high-dimensional feature spaces?

The good news is that Yes, sometimes it is possible to take advantage of high dimensional representations without actually having to work in the high dimensional space.

In certain cases, it is possible to compute dot products in these high-dimensional feature spaces without actually having to explicitly carry out the mapping into these spaces. If the subsequent processing can be carried out using dot products exclusively, then we can work in the high dimensional space *without ever explicitly mapping into the spaces*!

Slide 6

The Kernel Trick

Can we get around this problem and still get the benefit of high-D?

Yes!

Sometimes it is possible to compute dot-products without explicitly mapping into the high dimensional feature space. We employ dot products of the form

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle$$

which allow us to compute the value of the dot product in $\mathcal{H}$ without having to explicitly compute the map Φ.

Slide 7

The Kernel Trick

Given *any* algorithm that can be expresssed solely in terms of dot products, this kernel method allows us to construct different nonlinear versions of it.

Now, we go in search of an algorithm to try this trick on…

3 PCA

Now, taking that kernel idea and keeping it in the back of your mind, we move on to describe one kind of statistical anaylsis technique that, as we will see, can be expressed exclusively in terms of dot products. This technique is called Principal Components Analysis.

Principal Components Analysis (PCA) attempts to provide us with a set of orthogonal axes along which we can project our data, hopefully allowing us to account for most of the data with just the first few axes in teh new space.

Slide 8

PCA

Principal Components Analysis (PCA) attempts to efficiently represent the data by finding orthonormal axes which maximally decorrelate the data

Makes Following assumtions:

- Sources are Gaussian
- Sources are independent and stationary (iid)

In the following example, we consider a person who is attempting to efficiently report measurements of shoe-sizes. Clearly, there are a large number of possible measurements they could take. But were they to take a lot of data, and plot foot length vs. foot width, you'd see that the two are highly correlated. PCA would tell them that, should they want to describe shoe size with a single number, most of the information is conveyed by reporting the projection of a foot onto the line on the right. Then, the rest of the information, hould it be needed, is along the orthogonal axis.

Slide 9

PCA

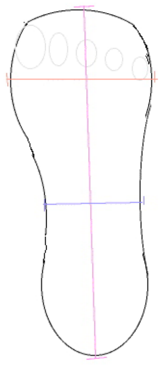
Principal Components Analysis (PCA) attempts to efficiently represent the data by finding orthonormal axes which maximally decorrelate the data

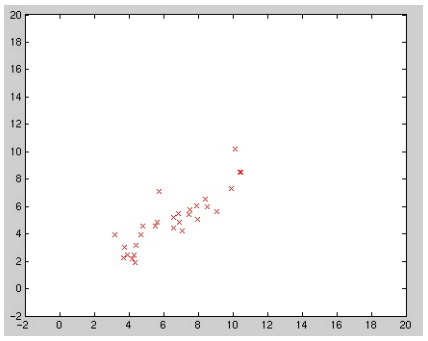
Makes Following assumtions:

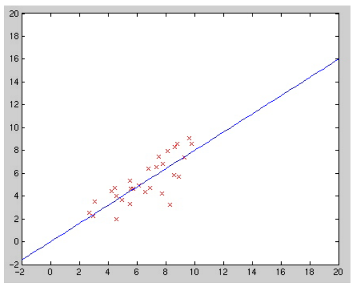
- Sources are Gaussian
- Sources are independent and stationary (iid)

Best shown with an example:

Slide 10







If we think of PCA as a method for blind source separation, it can be shown that, given the assumptions of gaussianity and iid, the principal components are the maximum likelihood estimators of the sources.

Slide 11

The Standard PCA Algorithm

Centered Observations: column vectors $x_i \in \mathbb{R}^N, i = 1, \ldots, m$
(Centered meaning: $\sum_{i=1}^m x_i = 0$)

PCA finds the principal axes by diagonalizing the covariance matrix

$$C = \frac{1}{m} \sum_{j=1}^{m} x_j x_j^\top \tag{5}$$

Note that C is positive definite, and thus can be diagonalized with nonnegative eigenvalues.

$$\lambda \boldsymbol{v} = C \boldsymbol{v} \tag{6}$$

Slide 12

Using PCA

1. Find eigenvectors, and arrange in order of decreasing eigenvalue.
2. Project test points onto eigenvectors
3. use those coefficients to do something useful (classification, image reconstruction, etc).

Slide 13

Rewriting PCA in terms of dot products

First, we need to remember that the eigenvectors lie in the span of $x_1 \dots x_n$ **Proof**: Substituting equation 4 into 5, we get

$$C\boldsymbol{v} = \frac{1}{m}\sum_{j=1}^{m} x_j x_j^{\mathsf{T}} \boldsymbol{v} = \lambda \boldsymbol{v}$$

Thus,

$$\begin{aligned} \boldsymbol{v} = & \tfrac{1}{m\lambda}\textstyle\sum_{j=1}^{m} x_j x_j^{\mathsf{T}} \boldsymbol{v} \\ = & \tfrac{1}{m\lambda}\textstyle\sum_{j=1}^{m} (x_j \cdot \boldsymbol{v}) x_j \end{aligned} \tag{7}$$

Slide 14

Show that $(\boldsymbol{x}\boldsymbol{x}^T)\boldsymbol{v} = (\boldsymbol{x}\cdot\boldsymbol{v})\boldsymbol{x}$

$$
\begin{aligned}
(\boldsymbol{x}\boldsymbol{x}^T)\boldsymbol{v} &= \begin{pmatrix} x_1x_1 & x_1x_2 & \dots & x_1x_M \\ x_2x_1 & x_2x_2 & \dots & x_2x_M \\ \vdots & \vdots & \ddots & \vdots \\ x_Mx_1 & x_Mx_2 & \dots & x_Mx_M \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_M \end{pmatrix} \\
&= \begin{pmatrix} x_1x_1v_1 + x_1x_2v_2 + \ldots + x_1x_Mv_M \\ x_2x_1v_1 + x_2x_2v_2 + \ldots + x_2x_Mv_M \\ \vdots \\ x_Mx_1v_1 + x_Mx_2v_2 + \ldots + x_Mx_Mv_M \end{pmatrix}
\end{aligned}
$$

Slide 15

$$
\begin{aligned}
&= \begin{pmatrix} (x_1v_1 + x_2v_2 + \ldots + x_Mv_M)\, x_1 \\ (x_1v_1 + x_2v_2 + \ldots + x_Mv_M)\, x_2 \\ \vdots \\ (x_1v_1 + x_2v_2 + \ldots + x_Mv_M)\, x_M \end{pmatrix} \\
&= \begin{pmatrix} x_1v_1 + x_2v_2 + \ldots + x_Mv_M \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{pmatrix} \qquad (8) \\
&= (\boldsymbol{x}\cdot\boldsymbol{v})\boldsymbol{x} \qquad \square
\end{aligned}
$$

Slide 16

So, from before, we had

$$\begin{aligned}\boldsymbol{v} =& \quad \tfrac{1}{m\lambda}\textstyle\sum_{j=1}^{m} x_j x_j^{\mathsf{T}} \boldsymbol{v} \\ =& \quad \tfrac{1}{m\lambda}\textstyle\sum_{j=1}^{m} (x_j \cdot \boldsymbol{v}) x_j\end{aligned}$$

But $(x_j \cdot \boldsymbol{v})$ is just a scalar, so this means that all solutions $\boldsymbol{v}$ with $\lambda \neq 0$ lie in the span of $x_1 \ldots x_m$, i.e.,

$$\boldsymbol{v} = \sum_{i=1}^{m} \alpha_i x_i \tag{9}$$

Slide 17

If we first send the data into another space,

$$\Phi : \mathcal{X} \to \mathcal{H}, \mathrm{x} \mapsto \Phi(\mathrm{x}) \tag{10}$$

Then, assuming we can center the data (i.e., $\sum_{k=1}^{m} \Phi(x_k) = 0$ – this is shown in the appendix), we can write the covariance matrix

$$C = \frac{1}{m} \sum_{j=1}^{m} \Phi(x_j) \Phi(x_j)^{\mathsf{T}} \tag{11}$$

Which can be diagonalized with nonnegative eigenvalues satisfying

$$\lambda \boldsymbol{V} = C \boldsymbol{V} \tag{12}$$

Slide 18

Now, we just showed that all solutions $\boldsymbol{v}$ with $\lambda \neq 0$ lie in the span of $Phi(x_1), \ldots, \Phi(x_m)$, that is

$$Cv = \lambda v = \lambda \sum_{i=1}^{m} \alpha_i \Phi(x_i) \tag{13}$$

Substituting (10) and (12) into (11), we get

$$\sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_j \Phi(x_i) K(x_i, x_j) = m\lambda \sum_{j=1}^{m} \alpha_j \Phi(x_i \tag{14}$$

where $K(x_i, x_j)$ is an inner-product kernel defined by

$$K(x_i, x_j) = \Phi(x_i)^{\mathsf{T}} \Phi(x_i) \tag{15}$$

Slide 19

To express the relationship entirely in terms of the inner-product kernel, we premultiply both sides by $\Phi(x_k)^{\mathsf{T}}$ and define

- the $m \times m$ matrix $\boldsymbol{K}$, called the kernel matrix, whose ij-th element is the inner-product kernel $K(x_i, x_j)$
- the $m \times 1$ vector $\boldsymbol{\alpha}$, whose jth element is the coefficient α_j

and can finally rewrite the expression as the eigenvalue problem

$$\boldsymbol{K\alpha} = \lambda \boldsymbol{\alpha} \tag{16}$$

If we chose Φ properly, we don't have to compute the map explicitly. We can use a kernel function

$$k(\Phi(x_i), \Phi(x_j)) = \langle \Phi(x_i), \Phi(x_j) \rangle \tag{17}$$

Which can potentially save us a lot of computation.

The final step is showing how to project a test point onto the eigenvectors in the high-dimensional space $\mathcal{H}$ in terms of a dot product so we can still keep on with the kernel trick.

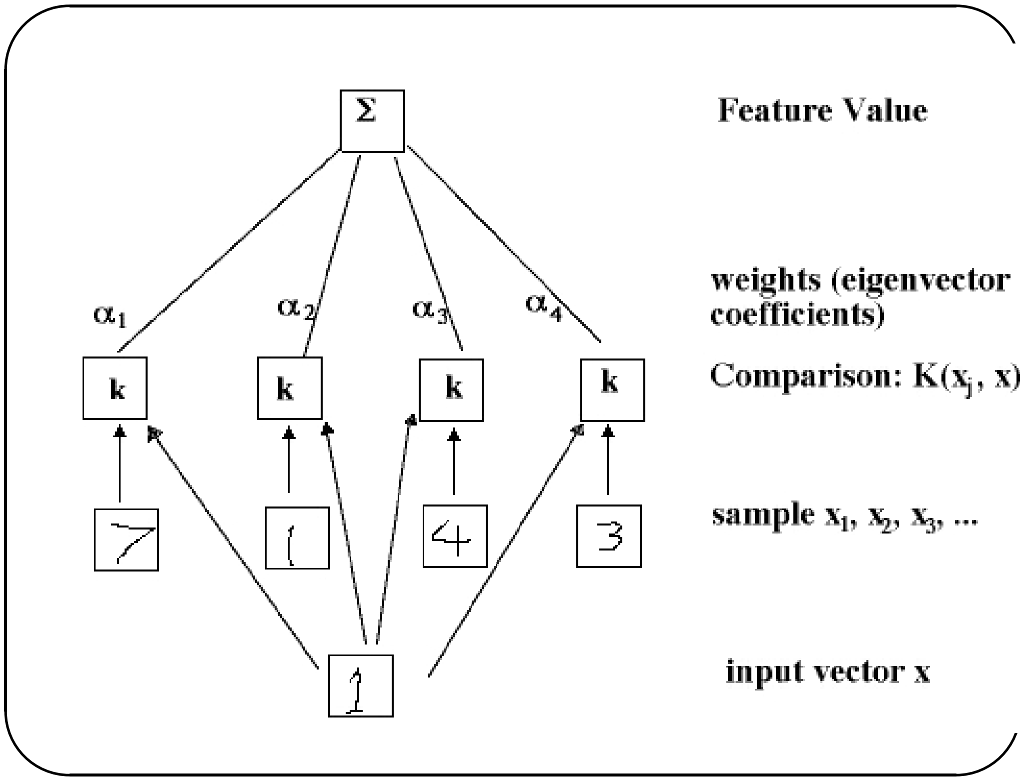
Slide 20

Extracting Principal Components

The resulting set of eigenvectors V^k in $\mathcal{H}$ are then used to extract the Principal Components of a test point by

$$\langle \boldsymbol{v}^k, \Phi(x) \rangle = \sum_{i=1}^{m} \alpha_i^n k(x_i, x), \quad n = 1, \ldots, p; \tag{18}$$

Slide 21

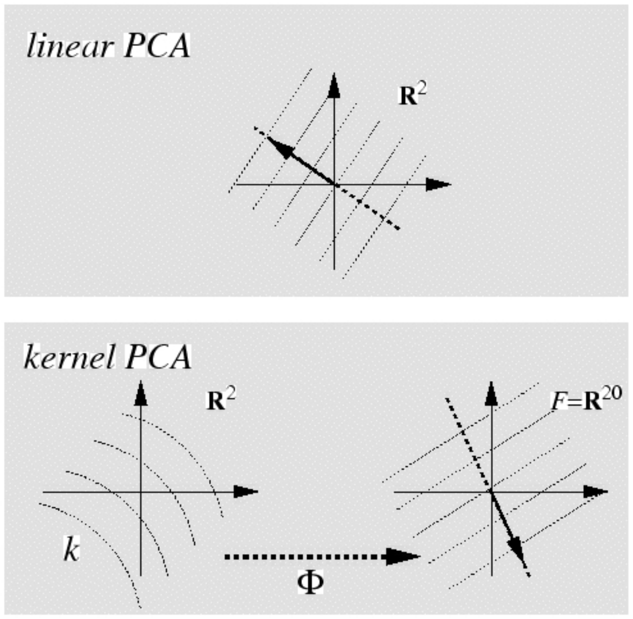


Having shown how to do kernel PCA, I now come to discussion of some of the differences betwen kernel PCA and linear PCA.

First off, what PCA gives us is a set of components in feature space which may not be easily interpretable in terms of the input space. Slide 22 shows one way of thinking about this in an example of 2-D inputs.

Slide 22

Kernel PCA vs regular (linear) PCA



Slide 23

Dimensionality reduction, etc.

Suppose that the number of observations m exceeds the input dimensionality n.

In linear PCA, we can find most n nonzero eigenvalues.

Kernel PCA can find up to m nonzero eigenvalues. Thus, this is not necessarily a dimensionality reduction.

Furthermore, it may not be possible to find an exact preimage in input space of a reconstructed pattern based on a few of the eigenvectors.

Another example might be face processing. In regular PCA, we can visualize the eigenfaces and have some intuition about them as capturing certain aspects of faceness. However, it may be more difficult to visualize kernel eigenfaces, because some directions in feature space might not have preimages in input space.

Slide 24

Computational Comp[lexity

If the kernel functions are easy to compute, e.g., polynomial kernels, the computational complexity of finding the eigenvalues is hardly changed by using K.

However, extracting principal components does take more work, because you have to evaluate the kernel function m times for each extracted principal component, instead of just doing one dot product as in linear PCA.

Although the added work of kernel PCA is small compared to the work it would take if we had to explicitly project each exemple and each subsequent test point into the high dimensional space, it still is a bit more costly than doing linear PCA. So we want to verify that we are compensated by improving performance or decreasing necessary complexity in subsequent stages of classification.

Slide 25

So, does it work?

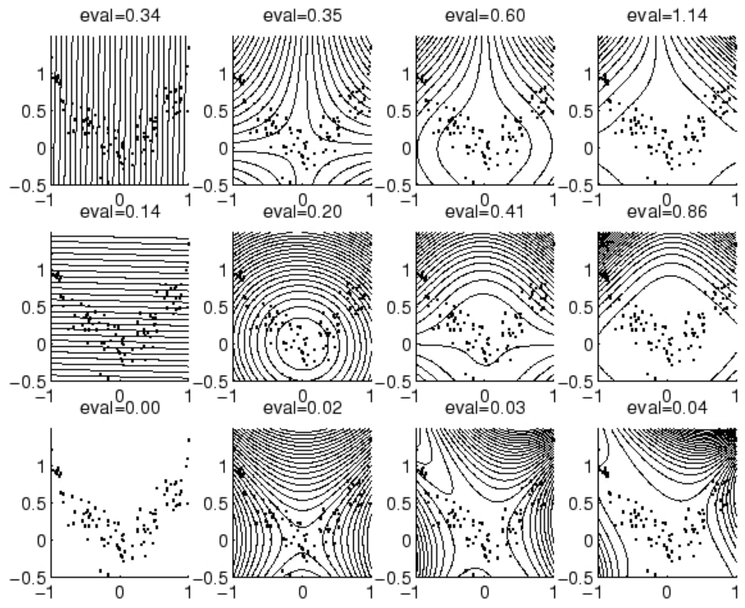
Examples:

- Toy examples
- Handwritten character recognition

In this first example, a parabola with gaussian noise added is shown. From right to left, the polynomial degree of the kernel increases from 1 to 4. From top to bottom: the first three eigenvectors are shown. Note that only two eigenvectors are available in linear PCA.

Slide 26

Example 1



Here, the data set is composed of three clusters. In this case, they are using a radial basis function (i.e., Gaussian) for the kernel. Note that the first two principal components nicely separate the data.

Slide 27

Example 2