# Visual Analytics
# Lecture week 11: Kernel Methods

Ian T. Nabney

University of BRISTOL

# Overview

We are working our way towards understanding Gaussian Processes for regression and the Gaussian Process Latent Variable Model as a method for dimensionality reduction. This will bring together many themes from the unit: dimensionality reduction, dissimilarity measures, and Bayesian inference.

- Understand fundamentals of dual representations
- Able to create kernel functions
- Understand basics of kernel PCA

Further reading: Bishop sections 6.1, 6.2, and 12.3.

# Context

- There is a class of pattern recognition techniques in which the training data points, or a subset of them, are kept and used also for prediction: e.g., the Parzen probability density model and the nearest neighbour classifier.
- Many linear parametric models can be re-cast into an equivalent 'dual representation' in which the predictions are also based on linear combinations of a symmetric kernel function evaluated at the training data points.
- This can be related to the use of dissimilarity functions in topographic dimensionality reduction.
- One of the most significant developments has been the extension of kernels to handle symbolic objects, thereby greatly expanding the range of problems that can be addressed.

# Dual representations

- Consider a linear regression model whose parameters are determined by minimizing a regularized sum-of-squares error function given by

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n) - t_n\right\}^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} \tag{1}$$

  where $\lambda \geqslant 0$.
- The solution for $\mathbf{w}$ is a linear combination of the vectors $\phi(\mathbf{x}_n)$, with coefficients that are functions of $\mathbf{w}$,

$$\mathbf{w} = -\frac{1}{\lambda}\sum_{n=1}^{N}\left\{\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n) - t_n\right\}\phi(\mathbf{x}_n) = \sum_{n=1}^{N} a_n\phi(\mathbf{x}_n) = \mathbf{\Phi}^{\mathrm{T}}\mathbf{a} \tag{2}$$

  where $\mathbf{\Phi}$ is the design matrix, whose $n^{\mathrm{th}}$ row is given by $\phi(\mathbf{x}_n)^{\mathrm{T}}$. We have defined

$$a_n = -\frac{1}{\lambda}\left\{\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n) - t_n\right\}. \tag{3}$$

- Instead of working with the parameter vector $\mathbf{w}$, we can now reformulate the least-squares algorithm in terms of the parameter vector $\mathbf{a}$, giving rise to a dual representation.

- Substitute $\mathbf{w} = \mathbf{\Phi}^{\mathrm{T}}\mathbf{a}$ into $J(\mathbf{w})$

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^{\mathrm{T}}\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}\mathbf{a} - \mathbf{a}^{\mathrm{T}}\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}\boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^{\mathrm{T}}\boldsymbol{t} + \frac{\lambda}{2}\mathbf{a}^{\mathrm{T}}\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}\mathbf{a} \tag{4}$$

  where $\boldsymbol{t} = (t_1, \ldots, t_N)^{\mathrm{T}}$.

- We now define the Gram matrix $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}$, which is an $N \times N$ symmetric matrix with elements

$$K_{nm} = \phi(\mathbf{x}_n)^{\mathrm{T}}\phi(\mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) \tag{5}$$

  where we have introduced the kernel function $k(\mathbf{x}, \mathbf{x}')$ defined by $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^{\mathrm{T}}\phi(\mathbf{x}')$.

- In terms of the Gram matrix, the sum-of-squares error function can be written as

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^{\mathrm{T}}\mathbf{K}\boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^{\mathrm{T}}\boldsymbol{t} + \frac{\lambda}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{a}. \tag{6}$$

  Setting the gradient of $J(\mathbf{a})$ with respect to $\mathbf{a}$ to zero, we obtain the following solution

$$\mathbf{a} = (\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\,\boldsymbol{t}. \tag{7}$$

# Dual formulation

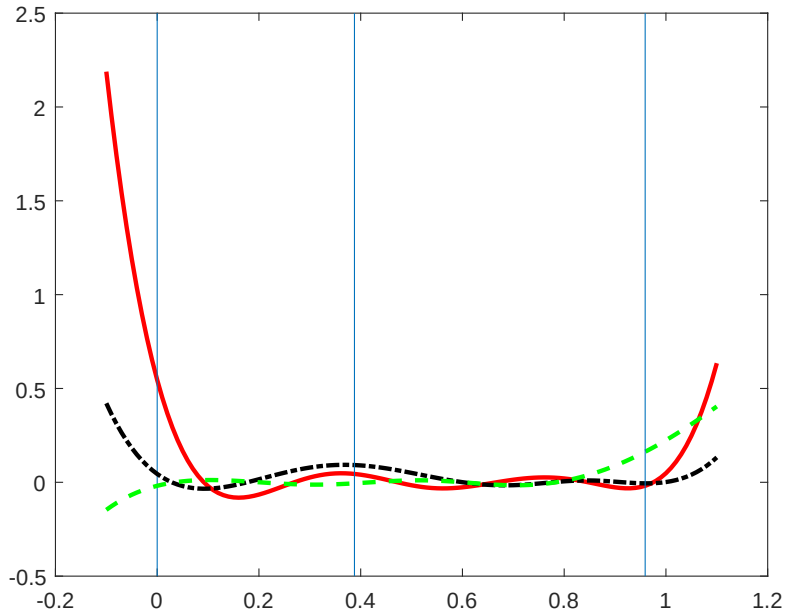
- We obtain the following prediction for a new input $\mathbf{x}$

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}) = \mathbf{a}^{\mathrm{T}}\mathbf{\Phi}\phi(\mathbf{x}) = \mathbf{k}(\mathbf{x})^{\mathrm{T}}\left(\mathbf{K} + \lambda\mathbf{I}_N\right)^{-1}\boldsymbol{t} \tag{8}$$

  where we have defined the vector $\mathbf{k}(\mathbf{x})$ with elements $k_n(\mathbf{x}) = k(\mathbf{x}_n, \mathbf{x})$.
- Thus the dual formulation allows the solution to the least-squares problem to be expressed entirely in terms of the kernel function $k(\mathbf{x}, \mathbf{x}')$.
- Note that the prediction at $\mathbf{x}$ is given by a linear combination of the target values from the training set.
- In the dual formulation, we determine the parameter vector $\mathbf{a}$ by inverting an $N \times N$ matrix, whereas in the original parameter space formulation we had to invert an $M \times M$ matrix in order to determine $\mathbf{w}$. Because $N$ is typically much larger than $M$, the dual formulation does not seem to be particularly useful.
- However, the advantage of the dual formulation is that it is expressed entirely in terms of the kernel function $k(\mathbf{x}, \mathbf{x}')$. We can therefore work directly in terms of kernels and avoid the explicit introduction of the feature vector $\phi(\mathbf{x})$, which allows us implicitly to use feature spaces of high, even infinite, dimensionality.

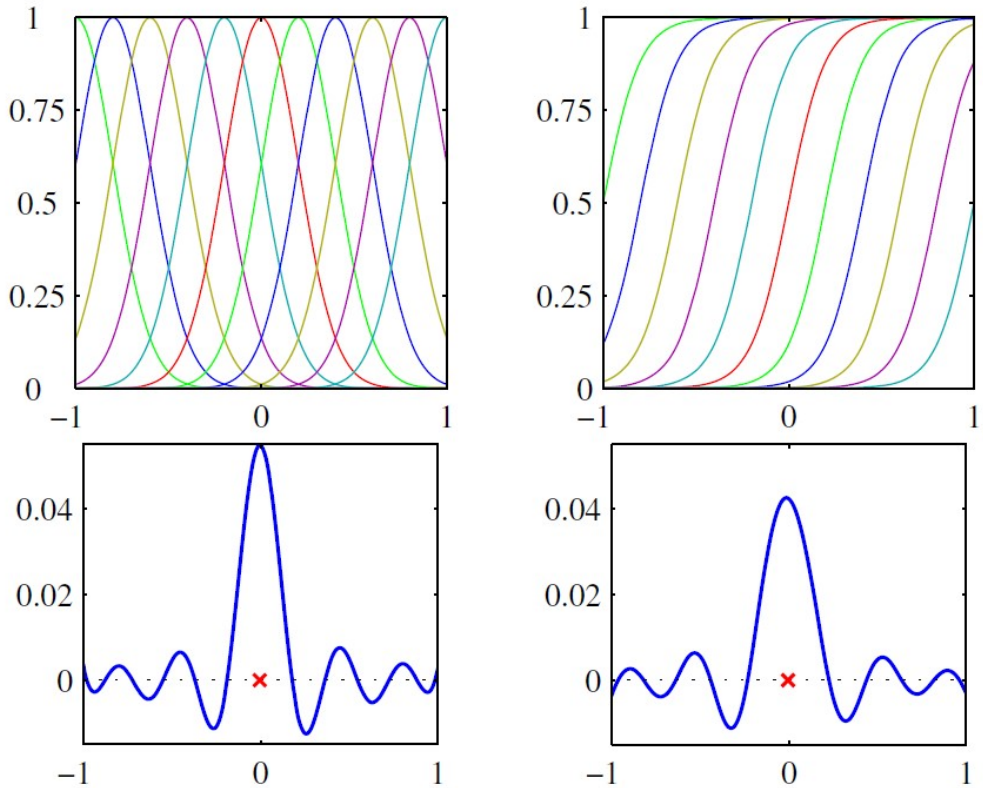
# Dual basis for RBF

- The RBF network is a linear combination of basis functions, so can we consider its dual formulation.
- In most cases the dual basis functions are localised around the data point that they correspond to, which explains how the network output can be localised, at least in the region of the training data.
- Dual basis functions for $r^4 \log r$ show localised damped oscillations in the training data range $[0, 1]$ on a sine function, while outside that region they grow rapidly.




Plotted for 1st (solid line), 20th (dash-dotted line) and 48th (dashed line) points in the dataset. The $x$-coordinate of the data points are given by the vertical lines from left to right.

# Constructing kernels




The lower plot shows the kernel function $k(x, x')$ defined by plotted as a function $k(x, x') = \boldsymbol{\phi}(x)^{\mathrm{T}}\boldsymbol{\phi}(x')$ of $x$ for $x' = 0$, while the upper plot shows the corresponding basis functions given by 'Gaussians' (left column), and logistic sigmoids (right column).

# Direct construction

- We must ensure that the function we choose is a valid kernel, in other words that it corresponds to a scalar product in some (perhaps infinite-dimensional) feature space.
- A necessary and sufficient condition for a function $k(\mathbf{x}, \mathbf{x}')$ to be a valid kernel is that the Gram matrix $\mathbf{K}$, whose elements are given by $k(\mathbf{x}_n, \mathbf{x}_m)$, should be positive semidefinite for all possible choices of the set $\{\mathbf{x}_n\}$.
- This means that all its singular values are non-negative.
- One powerful technique for constructing new kernels is to build them out of simpler kernels as building blocks (see notes).

# Kernel examples

- The kernel $k(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^{\mathrm{T}}\mathbf{x}' + c\right)^{M}$ with $c > 0$ contains all polynomials of degree up to $M$. For instance, if $\mathbf{x}$ and $\mathbf{x}'$ are two images, then the kernel represents a particular weighted sum of all possible products of $M$ pixels in the first image with $M$ pixels in the second image.
- Another commonly used kernel takes the form

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\|\mathbf{x} - \mathbf{x}'\|^2 / 2\sigma^2\right) \tag{9}$$

  and is often called a 'Gaussian' kernel (though it is not normalised).
- Kernel functions can be defined over objects as diverse as graphs, sets, strings, and text documents. Consider, for instance, a fixed set and define a nonvectorial space consisting of all possible subsets of this set. If $A_1$ and $A_2$ are two such subsets then

$$k(A_1, A_2) = 2^{|A_1 \cap A_2|} \tag{10}$$

  is a kernel.

# Kernel trick

- The concept of a kernel formulated as an inner product in a feature space allows us to build interesting extensions of many well-known algorithms by making use of the kernel trick, also known as kernel substitution.
- The general idea is that if we have an algorithm formulated in such a way that the input vector $\mathbf{x}$ enters only in the form of scalar products, then we can replace that scalar product with some other choice of kernel.
- For instance, the technique of kernel substitution can be applied to PCA to create a nonlinear variant.
- Other examples of kernel substitution include nearest-neighbour classifiers and the kernel Fisher discriminant.

# Kernel PCA

- Consider a data set $\{\mathbf{x}_n\}$ of observations, where $n = 1, \ldots, N$, in a space of dimensionality $D$. We shall assume that we have already subtracted the sample mean from each of the vectors $\mathbf{x}_n$.
- The first step is to express conventional PCA in such a form that the data vectors $\{\mathbf{x}_n\}$ appear only in the form of the scalar products $\mathbf{x}_n^{\mathrm{T}}\mathbf{x}_m$.
- The principal components are defined by the eigenvectors $\mathbf{u}_i$ of the covariance matrix

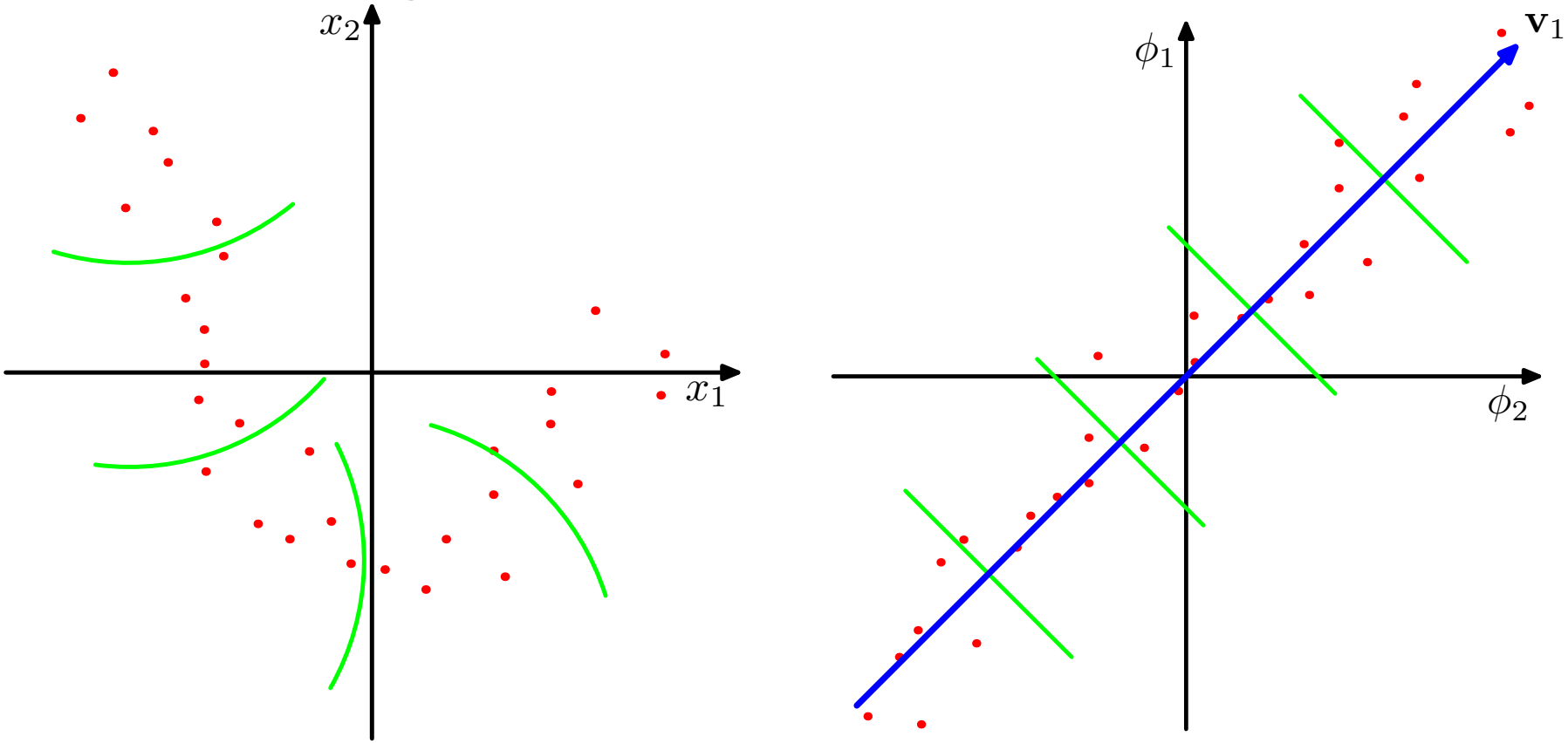
$$\mathbf{S}\mathbf{u}_i = \lambda_i \mathbf{u}_i \tag{11}$$

  where $i = 1, \ldots, D$.
- Here the $D \times D$ sample covariance matrix $\mathbf{S}$ is defined by

$$\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^{\mathrm{T}}, \tag{12}$$

  and the eigenvectors are normalized such that $\mathbf{u}_i^{\mathrm{T}}\mathbf{u}_i = 1$.

Consider a nonlinear transformation $\phi(\mathbf{x})$ into an $M$-dimensional feature space. We can now perform standard PCA in the feature space, which implicitly defines a nonlinear principal component model in the original data space.



# Solving the eigenproblem

- the eigenvector equations tell us that $\mathbf{v}_i$ satisfies

$$\frac{1}{N}\sum_{n=1}^{N}\phi(\mathbf{x}_n)\left\{\phi(\mathbf{x}_n)^{\mathrm{T}}\mathbf{v}_i\right\} = \lambda_i\mathbf{v}_i \tag{13}$$

and so we see that (provided $\lambda_i > 0$) the vector $\mathbf{v}_i$ is given by a linear combination of the $\phi(\mathbf{x}_n)$ and so can be written in the form

$$\mathbf{v}_i = \sum_{n=1}^{N} a_{in}\phi(\mathbf{x}_n). \tag{14}$$

- Substitute this expansion back into the eigenvector equation, and multiplying both sides by $\phi(\mathbf{x}_l)^{\mathrm{T}}$ gives

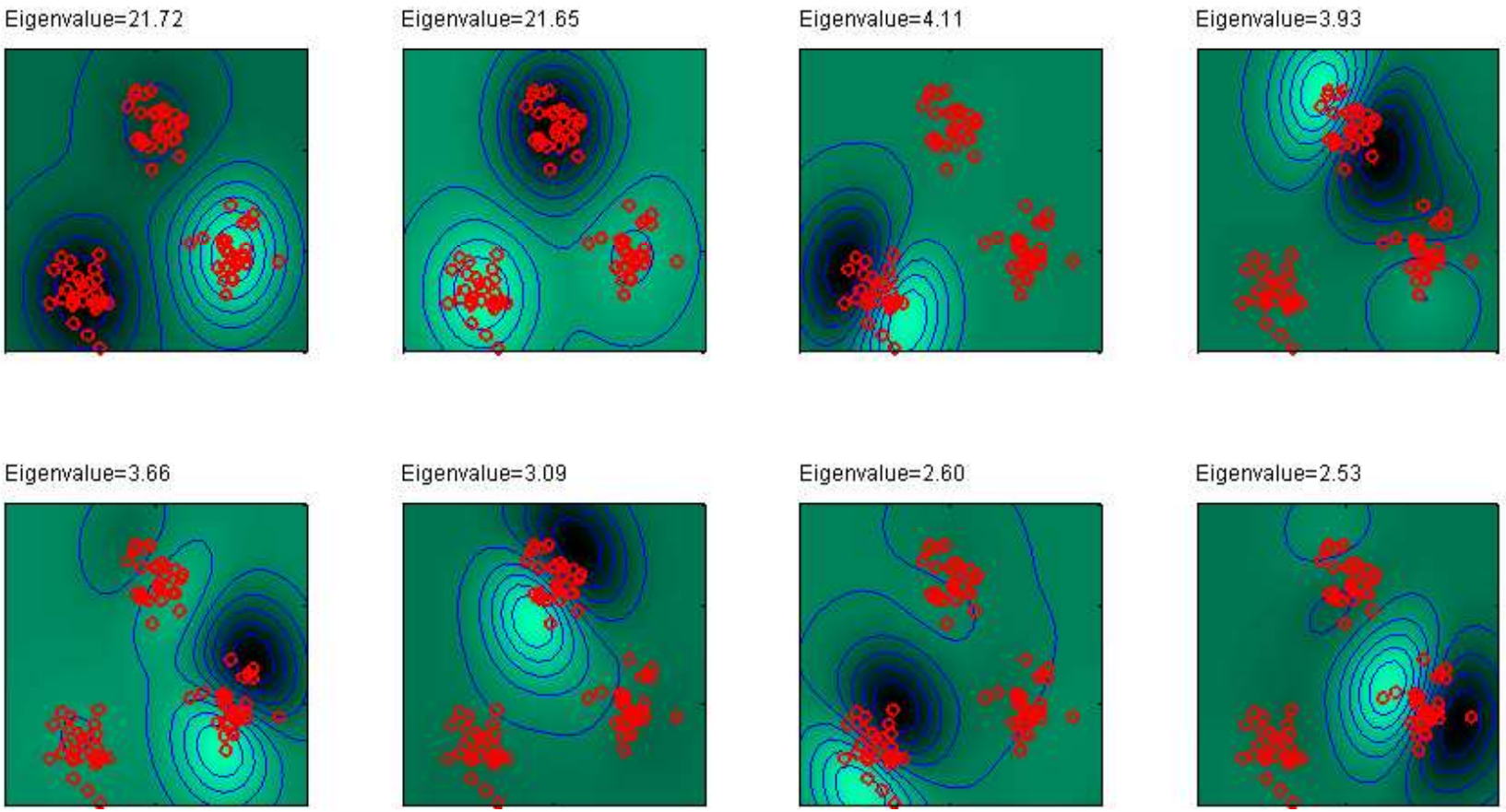
$$\mathbf{K}^2\mathbf{a}_i = \lambda_i N\mathbf{K}\mathbf{a}_i \tag{15}$$

- We can find solutions for $\mathbf{a}_i$ by solving the following eigenvalue problem

$$\mathbf{K}\mathbf{a}_i = \lambda_i N\mathbf{a}_i \tag{16}$$

in which we have removed a factor of $\mathbf{K}$ from both sides of (15). Note that the solutions of (15) and (16) differ only by eigenvectors of $\mathbf{K}$ having zero eigenvalues that do not affect the principal components projection.

# Kernel PCA example




Example of kernel PCA, with a Gaussian kernel applied to a synthetic data set in two dimensions, showing the first eight eigenfunctions along with their eigenvalues.

# Overview

We are working our way towards understanding Gaussian Processes for regression and the Gaussian Process Latent Variable Model as a method for dimensionality reduction.

- Understand fundamentals of dual representations
- Able to create kernel functions
- Understand basics of kernel PCA

The next lecture is about Gaussian Processes for regression.
A Python implementation of kernel PCA can be found at
`https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.KernelPCA.html`