Representations, kernels and Radial basis function networks

Kernel methods

Francesco Corona

Outline

Kernel methods

Kernel methods

We have considered linear parametric models for regression and classification

- The form of the mapping $y(\mathbf{x}, \mathbf{w})$ from input $\mathbf{x}$ to output $\mathbf{y}$ is governed by a vector $\mathbf{w}$ of adaptive parameters

During learning, a set of training data is used either to obtain a point estimate of the parameter vector or to determine a posterior distribution over this vector

- The training data is then discarded, and predictions for new inputs are based purely on the learned parameter vector $\mathbf{w}$

The approach is also used in nonlinear parametric models, like neural networks

Kernel methods (cont.)

However, there is a class of techniques, in which the training data points, or a subset of them, are kept and used also during the prediction phase

- Parzen probability density model comprised a linear combination of *kernel* functions each one centred on one of the training data points
- Nearest neighbours for classification involved assigning to each test vector the same label as the closest example from the training set

These are examples of **memory-based methods** that involve storing the entire training set in order to make predictions for future data points

- Require a metric to be defined that measures a similarity of any two vectors in the input space
- Fast to *train* but slow at making predictions for test data points

Kernel methods (cont.)

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 kernel function evaluated at training points

For models which are based on a fixed nonlinear **feature space** mapping $\phi(\mathbf{x})$, the **kernel function** is given by the relationship

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}') \tag{1}$$

The kernel is a symmetric function of its arguments, $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$

The simplest example of a kernel function is obtained by considering the identity mapping $\phi(\mathbf{x}) = \mathbf{x}$ that is $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$ $\longleftarrow$ **linear kernel**

Kernel methods (cont.)

The concept of a kernel formulated as an inner product in a feature space allows to build extensions of a large number of very 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

There are numerous forms of kernel functions in common use

Kernel methods (cont.)

Many have the property of being a function of the difference between arguments

$$k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x} - \mathbf{x}')$$

known as **stationary kernels** because invariant to translations in input space

Others **homogeneous kernels**, or **radial basis functions**, which depend only on the magnitude of the distance (typically Euclidean) between arguments

$$k(\mathbf{x}, \mathbf{x}') = k(||\mathbf{x} - \mathbf{x}'||)$$

Dual representation
Kernel methods

Dual representation

Many linear models for regression and classification can be re-formulated in terms of a **dual representation** in which the kernel function arises naturally

Consider a linear regression model whose parameters are determined by minimising the regularised sum-of-squares error function given by

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(\mathbf{w}^T\phi(\mathbf{x}_n) - t_n\right)^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}, \quad \text{with } \lambda \geq 0 \tag{2}$$

If we set the gradient of $J(\mathbf{w})$ wrt $\mathbf{w}$ equal to zero, we get the solution

$$\mathbf{w} = -\frac{1}{\lambda}\sum_{n=1}^{N}\left(\mathbf{w}^T\phi(\mathbf{x}_n) - t_n\right)\phi(\mathbf{x}_n) = \sum_{n=1}^{N} a_n\phi(\mathbf{x}_n) = \mathbf{\Phi}^T\mathbf{a} \tag{3}$$

- $\mathbf{\Phi}$ is the design matrix, in which the n-th row is $\phi(\mathbf{x}_n)^T$
- $\mathbf{a} = (a_1, \ldots, a_N)^T$, with $a_n = -1/\lambda\left(\mathbf{w}^T\phi(\mathbf{x}_n) - t_n\right)$

A linear combination of vectors $\phi(\mathbf{x}_n)$ with coefficients functions of $\mathbf{w}$

Dual representation (cont.)

In a **dual representation** instead of working with a parameter vector $\mathbf{w}$, we re-formulate the least-squares algorithm in terms of the parameter vector $\mathbf{a}$

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

where we substituted $\mathbf{w} = \mathbf{\Phi}^T\mathbf{a}$ into $J(\mathbf{w})$ and with $\mathtt{t} = (t_1, \ldots, t_n)^T$

We define the $N \times N$ symmetric **Gram matrix** $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^T$ with elements

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

evaluated using the **kernel function** $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T\phi(\mathbf{x}')$

Dual representation (cont.)

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^T \underbrace{\mathbf{\Phi}\mathbf{\Phi}^T}_{\mathbf{K}} \underbrace{\mathbf{\Phi}\mathbf{\Phi}^T}_{\mathbf{K}} \mathbf{a} - \mathbf{a}^T \underbrace{\mathbf{\Phi}\mathbf{\Phi}^T}_{\mathbf{K}} \mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^T \underbrace{\mathbf{\Phi}\mathbf{\Phi}^T}_{\mathbf{K}} \mathbf{a}$$

In terms of the Gram matrix, the sum-of-squares error function becomes

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

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

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

Dual representation (cont.)

And substituting back into the linear regression model, we obtain the following

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

as the prediction for a new input $\mathbf{x}$, with vector $\mathbf{k}(\mathbf{x}) = \Big(k(\mathbf{x}_1, \mathbf{x}), \ldots, k(\mathbf{x}_n, \mathbf{x} \Big)^T$

The prediction at $\mathbf{x}$ is a linear combination of target values in the training set

Dual representation (cont.)

In the dual formulation, we determine parameter $\mathbf{a}$ by inverting an $N \times N$ matrix

$$\mathbf{a} = (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \mathbf{t}$$

In the original formulation, we have to invert an $M \times M$ matrix to determine $\mathbf{w}$

$$\mathbf{w} = \left(\lambda \mathbf{I} + \mathbf{\Phi}^T \mathbf{\Phi}\right)^{-1} \mathbf{\Phi}^T \mathbf{t}$$

Typically $N \gg M$, the dual formulation does not seem to be of much use

- The advantage is that the dual formulation is expressed entirely in terms of the kernel function $k(\mathbf{x}, \mathbf{x}')$
- We can work directly in terms of kernels and avoid the explicit introduction of the feature vector $\phi(\mathbf{x})$

We can implicitly use feature spaces of high, even infinite, dimensionality

Constructing kernels
Kernel methods

Constructing kernels

To exploit kernel tricks, we need to be able to construct valid kernel functions

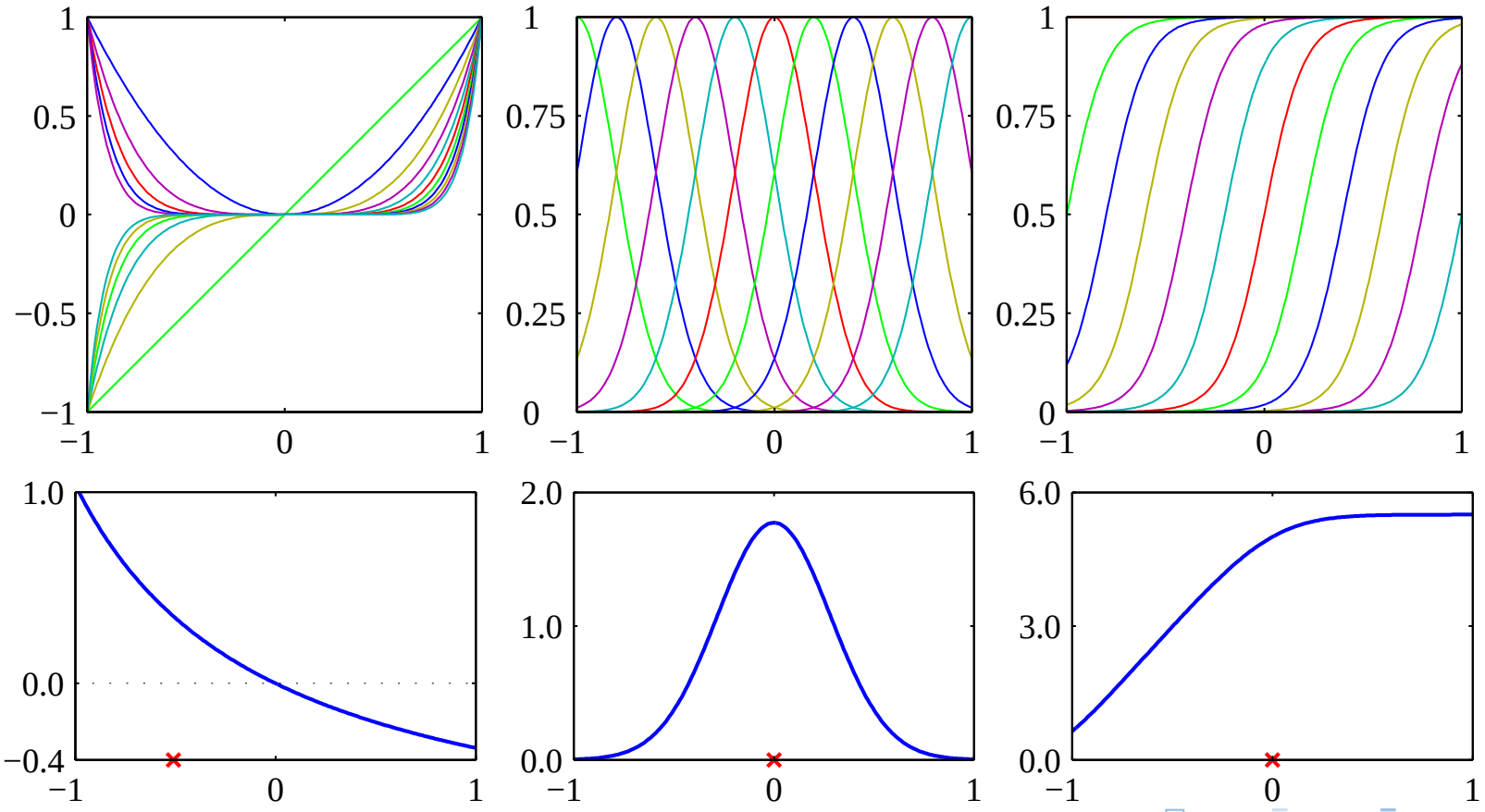
One approach is to first choose a feature space mapping $\phi(\mathbf{x})$

- Then use this to find the corresponding kernel

Constructing kernels (cont.)

For a 1-D input space, the kernel function with basis functions $\phi_i(x)$ is defined

$$k(x, x') = \phi(x)^T \phi(x) = \sum_{i=1}^{M} \phi_i(x)\phi_i(x') \tag{9}$$



Constructing kernels (cont.)

An alternative approach is to construct kernel functions directly

- we must ensure that the function we choose is a valid kernel
- it must correspond to a scalar product in some feature space

Constructing kernels (cont.)

As an example consider a 2-D space and the kernel function given by

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z})^2 \tag{10}$$

Because $\mathbf{x} = (x_1, x_2)$ and $\mathbf{z} = (z_1, z_2)$, we can expand the terms to get

$$\begin{aligned} k(\mathbf{x}, \mathbf{z}) &= (\mathbf{x}^T \mathbf{z})^2 = (x_1 z_1 + x_2 z_2)^2 \\ &= x_1^2 z_1^2 + 2 x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\ &= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)(z_1^2, \sqrt{2} z_1 z_2, z_2^2)^T \\ &= \phi(\mathbf{x})^T \phi(\mathbf{z}) \end{aligned} \tag{11}$$

which identifies the nonlinear feature mapping $\phi(\mathbf{x}) = (x_1^2, \sqrt{2} x_1 x_2, x_2^2)^T$

Constructing kernels (cont.)

We would need a simple way to test whether a function defines a valid kernel

- without having to construct the function $\phi(\mathbf{x})$ explicitly

A necessary and sufficient condition for a function $k(\mathbf{x}, \mathbf{x}')$ to be a valid kernel:

- 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 $\{\mathbf{x}_n\}$

One powerful technique for constructing new kernels is to build them out of simpler kernels as building blocks

- This can be done using some known properties

Constructing kernels (cont.)

Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$, the following kernels are valid too

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= c k_1(\mathbf{x}, \mathbf{x}'), \text{ with } c > 0 && (12)\\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}'), \text{ with } f(\cdot) \text{ any function} && (13)\\
k(\mathbf{x}, \mathbf{x}') &= q(k_1(\mathbf{x}, \mathbf{x}')), \text{ with } q(\cdot) \text{ a polynomial w/ coeffs} > 0 && (14)\\
k(\mathbf{x}, \mathbf{x}') &= \exp(k_1(\mathbf{x}, \mathbf{x}')) && (15)\\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') && (16)\\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}') && (17)\\
k(\mathbf{x}, \mathbf{x}') &= k_3(\phi(\mathbf{x}), \phi(\mathbf{x}')), \text{ with } \phi(\mathbf{x}) : \mathbf{x} \to \mathbb{R}^M \text{ and } k_3(\cdot, \cdot) \text{ in } \mathbb{R}^M && (18)\\
k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^T \mathbf{A} \mathbf{x}, \text{ with } \mathbf{A} \text{ symmetric PSD} && (19)\\
k(\mathbf{x}, \mathbf{x}') &= \cdots && (20)
\end{aligned}
$$

Constructing kernels (cont.)

Equipped with these properties, we can construct more complex kernels appropriate to specific applications

We require that the kernel $k(\mathbf{x}, \mathbf{x}')$ be symmetric and positive semidefinite and that it expresses the appropriate similarity between $\mathbf{x}$ and $\mathbf{x}'$ for the application

Constructing kernels (cont.)

Polynomial kernel: $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x})^2$ contains only second order terms

The more general kernel $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x} + c)^2$ with $c > 0$, the corresponding feature mapping $\phi(\mathbf{x})$ contains constant and linear terms, and order two ones

Similarly, $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x})^M$ contains all monomials of order M

Gaussian kernel: $k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right)$ is valid kernel

- The corresponding feature vector is infinite dimensional

Extension to inputs that are symbolic, rather than just vectors of real numbers

Kernel functions can be defined over objects as diverse as graphs, sets, strings

Constructing kernels (cont.)

One can consider a fixed set and define a space of all possible subsets of this set

If A_1 and A_2 are two such subsets then one simple choice of kernel would be

$$k(A_1, A_2) = 2^{|A_1 \cap A_2|} \tag{21}$$

$A_1 \cap A_2$ is intersection of sets A_1 and A_2, and $|A|$ is number of elements in A

A valid kernel function for it corresponds to an inner product in a feature space

Radial basis function networks
Kernel methods

Radial basis function networks

We saw regression models based on linear combinations of fixed basis functions

- We did not discuss in detail what form those basis functions might take

One widely used choice is radial basis functions, in which each basis function depends only on some kind radial distance (often Euclidean) from a centre $\boldsymbol{\mu}_j$

$$\phi_j(\mathbf{x}) = h\Big(||\mathbf{x} - \boldsymbol{\mu}_j||\Big)$$

Radial basis function networks (cont.)

Radial basis functions were introduced for exact function interpolation

- For a set of input vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and corresponding target values $\{t_1, \ldots, t_N\}$, find a smooth function $f(\mathbf{x})$ that fits the target values exactly
- That is, $f(\mathbf{x}_n) = t_n$, for $n = 1, \ldots, N$

This is done by expressing $f(\mathbf{x})$ as linear combination of radial basis functions

- One centred on every data point

$$f(\mathbf{x}) = \sum_{n=1}^{N} w_n h\Big(||\mathbf{x} - \mathbf{x}_n||\Big) \tag{22}$$

The values of the coefficients $\{w_n\}$, as many as constraints, can be found by least-squares, which leads to an undesirable perfect fit corresponding to overfit

Radial basis function networks (cont.)

Another possible interpretation comes from the case where the inputs are noisy

- If the additive input noise is described by a variable ξ with distribution $\nu(\xi)$, then the sum-of-squares error function takes the form

$$E = \frac{1}{2}\sum_{n=1}^{N}\int \Big(y(\mathbf{x}+\xi) - t_n\Big)^2 \nu(\xi)d\xi \tag{23}$$

And, by optimising with respect to the function $y(\mathbf{x})$, we obtain

$$y(\mathbf{x}) = \sum_{n=1}^{N} t_n h(\mathbf{x}-\mathbf{x}_n) \tag{24}$$

where the basis functions $h(\mathbf{x}-\mathbf{x}_n)$ are then given by the form

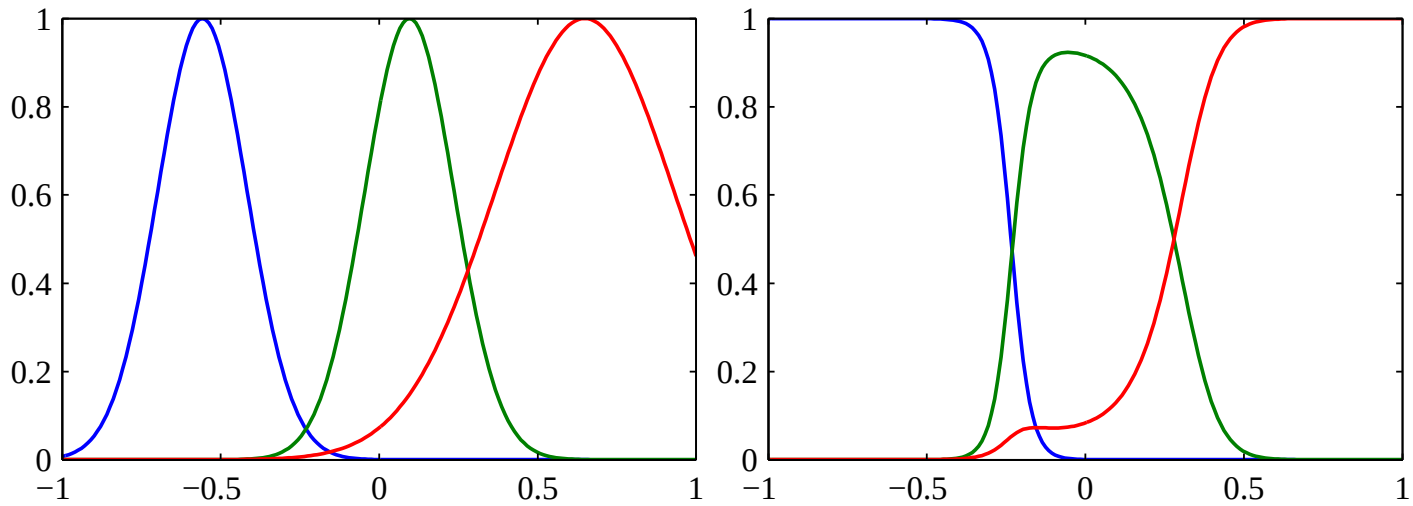
$$h(\mathbf{x}-\mathbf{x}_n) = \frac{\nu(\mathbf{x}-\mathbf{x}_n)}{\sum_{n=1}^{N}\nu(\mathbf{x}-\mathbf{x}_n)} \tag{25}$$

Radial basis function networks (cont.)

$$h(\mathbf{x} - \mathbf{x}_n) = \frac{\nu(\mathbf{x} - \mathbf{x}_n)}{\sum_{n=1}^{N} \nu(\mathbf{x} - \mathbf{x}_n)}$$

There is one basis function centred on every point and if the noise distribution is isotropic (it depends only on $||\xi||$), then the basis functions will be radial

Also, note that the basis functions are normalised, so that $\sum_n h(\mathbf{x} - \mathbf{x}_n) = 1$



Nadaraya-Watson
Radial basis function networks

Nadaraya-Watson

The prediction of a linear regression model for a new input $\mathbf{x}$ takes the form

$$y(\mathbf{x}, \mathbf{m}_N) = \sum_{n=1}^{N} \underbrace{\beta \phi(\mathbf{x})^T \mathbf{S}_N \phi(\mathbf{x}_n)}_{k(\mathbf{x}, \mathbf{x}_n)} t_n$$

A linear combination of the training set target values with coefficients given by

$$k(\mathbf{x}, \mathbf{x}') = \beta \phi(\mathbf{x})^T \mathbf{S}_N \phi(\mathbf{x}')$$

The equivalent kernel $k(\mathbf{x}, \mathbf{x}')$ is such that it satisfies the summation constraint

$$\sum_{n=1}^{N} k(\mathbf{x}, \mathbf{x}') = 1, \quad \forall \mathbf{x}$$

Nadaraya-Watson (cont.)

This view of linear basis functions models results from the Bayesian treatment

- We introduced a prior probability distribution over the model parameters $\mathbf{w}$

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{m}_0, \mathbf{S}_0)$$

- The prior is a Gaussian distribution, as this conjugates with the likelihood

$$p(\mathrm{t}|\mathbf{w}) = \prod_{n=1}^{N} \mathcal{N}\left(t_n|\mathbf{w}^T\phi(\mathbf{x}_n), \beta\right)$$

- A result of iid samples for a target variable t with additive Gaussian noise

$$p(t|x, \mathbf{w}, \beta) = \mathcal{N}\left(t\middle|y(x, \mathbf{w}), \beta^{-1}\right)$$

The posterior distribution is $\propto$ to the product of likelihood function and prior

Nadaraya-Watson (cont.)

Due to the choice of a conjugate prior, the posterior distribution is Gaussian

$$p(\mathbf{w}|\mathrm{t}) = \mathcal{N}(\mathbf{w}|\mathbf{m}_N, \mathbf{S}_N) \quad \text{with} \quad \begin{cases} \mathbf{m}_N = \mathbf{S}_N(\mathbf{S}_0^{-1}\mathbf{m}_0 + \beta\mathbf{\Phi}^T\mathrm{t}) \\ \mathbf{S}_N^{-1} = \mathbf{S}_0^{-1} - \beta\mathbf{\Phi}^T\mathbf{\Phi} \end{cases}$$

Choosing a zero-mean isotropic Gaussian prior $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})$ then

$$p(\mathbf{w}|\mathrm{t}) = \mathcal{N}(\mathbf{w}|\mathbf{m}_N, \mathbf{S}_N) \quad \text{with} \quad \begin{cases} \mathbf{m}_N = \beta\mathbf{S}_N\mathbf{\Phi}^T\mathrm{t} \\ \mathbf{S}_N^{-1} = \alpha\mathbf{I} + \beta\mathbf{\Phi}^T\mathbf{\Phi} \end{cases}$$

Substituting $\mathbf{m}_N = \beta\mathbf{S}_N\mathbf{\Phi}^T\mathrm{t}$ into $y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x})$, we get

$$y(\mathbf{x}, \mathbf{m}_N) = \mathbf{m}_N^T\phi(\mathbf{x}) = \beta\phi(\mathbf{x})^T\mathbf{S}_N\mathbf{\Phi}^t\mathrm{t} = \sum_{n=1}^{N} \underbrace{\beta\phi(\mathbf{x})^T\mathbf{S}_N\phi(\mathbf{x}_n)}_{k(\mathbf{x},\mathbf{x}_n)} t_n$$

Nadaraya-Watson (cont.)

$$y(\mathbf{x}, \mathbf{m}_N) = \sum_{n=1}^{N} k(\mathbf{x}, \mathbf{x}_n) t_n$$

This kernel regression model can be seen also from a different perspective

Suppose we have a training set $\{\mathbf{x}_n, t_n\}$ and use a Parzen density estimator to model the joint distribution $p(\mathbf{x}, t)$

$$p(\mathbf{x}, t) = \frac{1}{N} \sum_{n=1}^{N} f(\mathbf{x} - \mathbf{x}_n, t - t_n) \tag{26}$$

where $f(\mathbf{x}, t)$ is the component density function, and there is one such component centred on each data point $(\mathbf{x}_n, t_n)$

Nadaraya-Watson (cont.)

We now find an expression for the regression function $y(\mathbf{x})$, corresponding to the conditional average of the target value conditioned on the input variable

$$\begin{aligned} y(\mathbf{x}) &= \mathbb{E}[t|\mathbf{x}] = \int_{-\infty}^{+\infty} t p(t|\mathbf{x}) \\ &= \frac{\int t p(\mathbf{x}, t) dt}{\int p(\mathbf{x}, t) dt} \\ &= \frac{\sum_{n=1}^{N} \int t f(\mathbf{x} - \mathbf{x}_n, t - t_n) dt}{\sum_{m=1}^{N} \int f(\mathbf{x} - \mathbf{x}_m, t - t_m) dt} \end{aligned} \tag{27}$$

We assume for simplicity that the component density function have zero mean

$$\int_{-\infty}^{+\infty} f(\mathbf{x}, t) t dt = 0, \quad \text{for all values of } \mathbf{x} \tag{28}$$

Nadaraya-Watson (cont.)

Using the change of variable $g(\mathbf{x}) = \int_{-\infty}^{+\infty} f(\mathbf{x}, t)dt$, we then obtain the form

$$y(\mathbf{x}) = \frac{\sum_{n=1}^{N} g(\mathbf{x} - \mathbf{x}_n)t_n}{\sum_{m=1}^{N} g(\mathbf{x} - \mathbf{x}_m)} = \sum_{n=1}^{N} k(\mathbf{x}, \mathbf{x}_n)t_n \tag{29}$$

which is referred to as the **Nadaraya-Watson** or the **kernel regression model**

The kernel function is given by

$$k(\mathbf{x}, \mathbf{x}_n) = \frac{g(\mathbf{x} - \mathbf{x}_n)}{\sum_{m=1}^{N} g(\mathbf{x} - \mathbf{x}_m)} \tag{30}$$

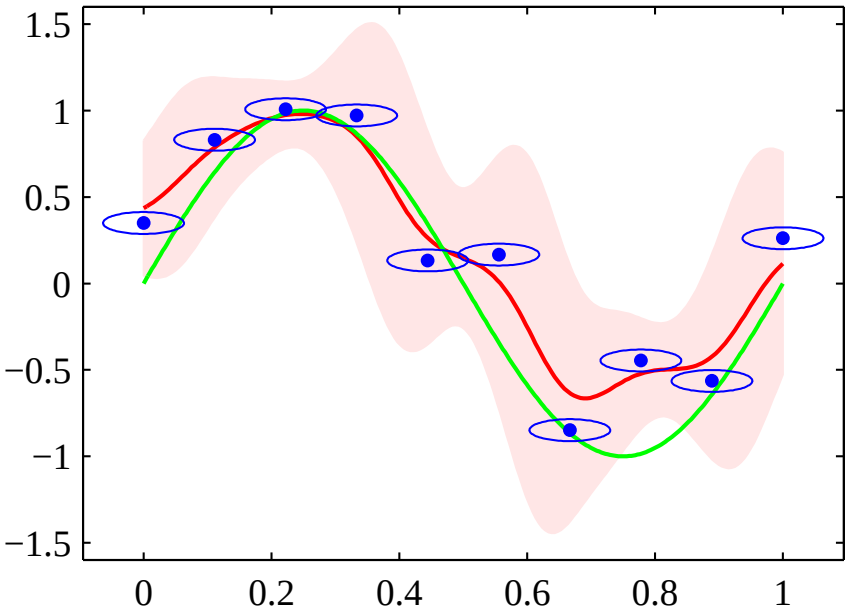
It satisfies the sum constraint

$$\sum_{n=1}^{N} k(\mathbf{x}, \mathbf{x}_n) = 1 \tag{31}$$

Nadaraya-Watson (cont.)

Consider the usual sine function (in green) over a single input x and an $f(x, t)$ given by a zero-mean isotropic Gaussian with variance σ^2 over $\mathbf{z} = (x, t)$

Data points are shown in blue and the red line shows the NW kernel regression



Each point is the centre of a isotropic Gaussian kernel, with one standard deviation contour

- Contours should be circular

The conditional mean $\mathbb{E}[t|x]$ of the conditional distribution $p(t|x)$ with two standard deviation bands is the regression function $y(\mathbf{x})$