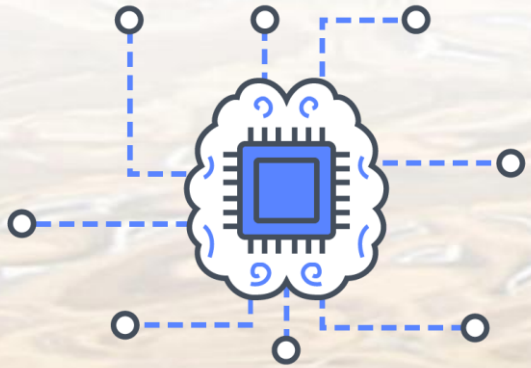


Machine Learning

Chapter 6: Kernel Methods

林嘉文 (Chia-Wen Lin)

清華大學電機系

cwlin@ee.nthu.edu.tw

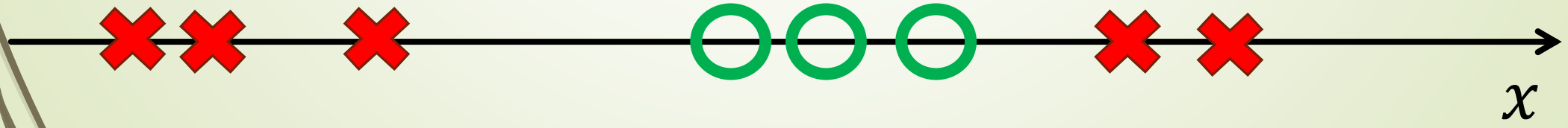
Training Data: Keep or Discard?

- **Parametric methods** (linear/nonlinear) so far:
  - Learn parameter vector $\mathbf{w}$ or posterior distribution $p(\mathbf{w}|\mathcal{D})$
  - Discard training data $\mathcal{D}$
- **Non-parametric methods**:
  - Parzen probability density model: set of kernel functions centered on training data points
  - Nearest neighbors technique: closest example(s) from the training set
  - Memory-based methods: similar examples from the training set
- **Kernel methods**:
  - Prediction is based on linear combinations of a kernel function evaluated at the training data points

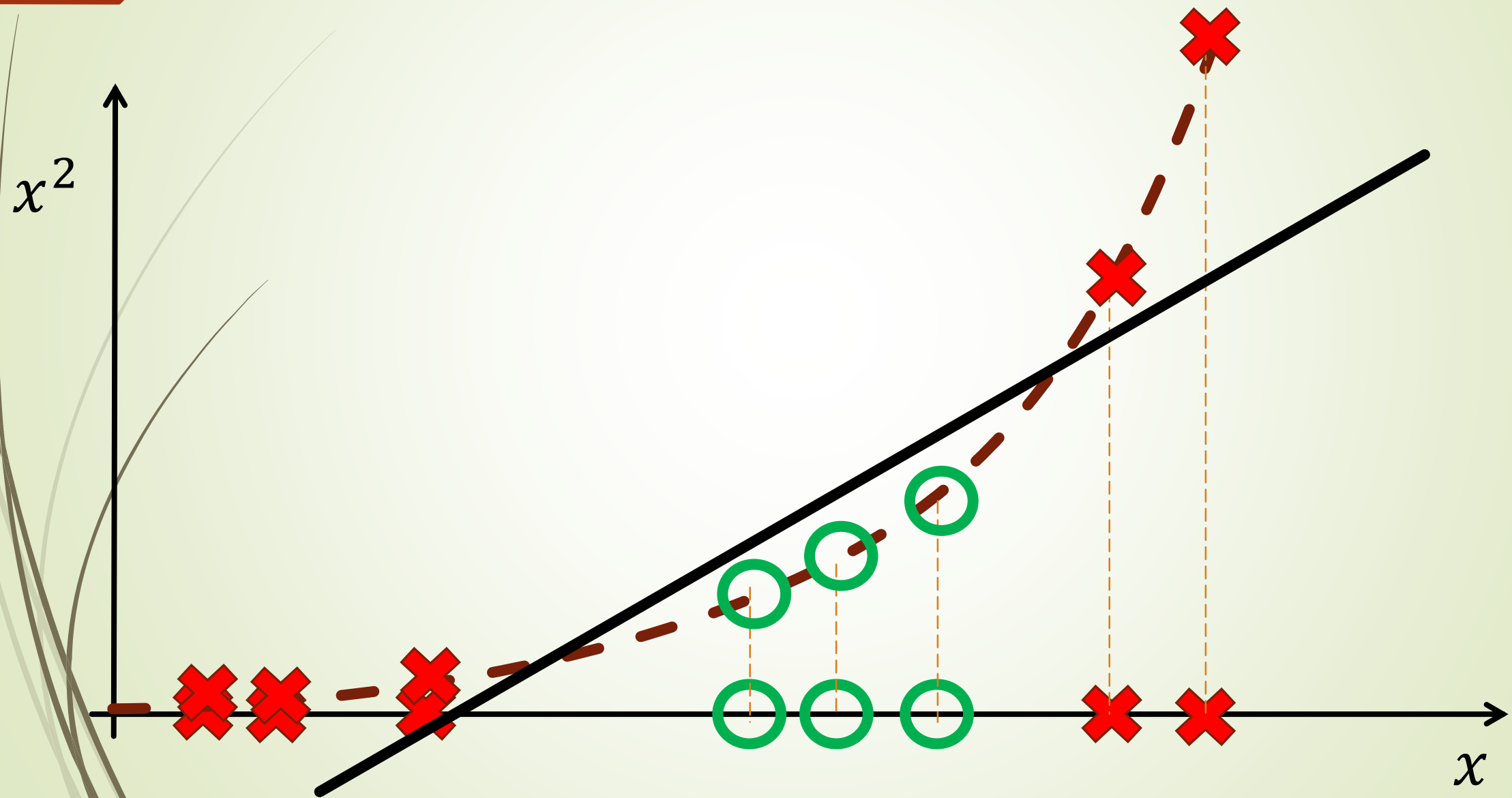
Non-linear Classification

- How do we separate the two classes using a hyperplane?

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \boldsymbol{\theta}^{\mathsf{T}}\mathbf{x} \geq 0 \\ 0 & \text{if } \boldsymbol{\theta}^{\mathsf{T}}\mathbf{x} < 0 \end{cases}$$



Non-linear Classification



Kernel Functions

- For models based on feature space mapping $\boldsymbol{\phi}(\mathbf{x})$:

$$k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{x}')$$

- Symmetric function: $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$
- simple example - linear kernel: $\boldsymbol{\phi}(\mathbf{x}) = \mathbf{x}$
- stationary kernel: $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x} - \mathbf{x}')$
- homogeneous kernel: $k(\mathbf{x}, \mathbf{x}') = k(\|\mathbf{x} - \mathbf{x}'\|)$

Algorithm, expressed in terms of scalar products can be reformulated using kernel substitution: PCA, nearest-neighbor classifiers, Fisher discriminant

Dual Representation

Many linear models for regression and classification can be reformulated in terms of a dual representation in which the kernel function arises naturally.

- Linear regression model ($\lambda \geq 0$):

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

- Set the gradient to zero:

$$\boldsymbol{\Phi}^{\mathsf{T}} = [\boldsymbol{\phi}(\mathbf{x}_1) \quad \cdots \quad \boldsymbol{\phi}(\mathbf{x}_N)]$$ **(design matrix)**

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

- Substitute $\mathbf{w}$ and define the **Gram matrix** $\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathsf{T}}$

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

Solution for Dual Problem

- In terms of new parameter vector $\mathbf{a}$:

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

- Set the gradient to zero:

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

- Prediction for a new input $\mathbf{x}$

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

where

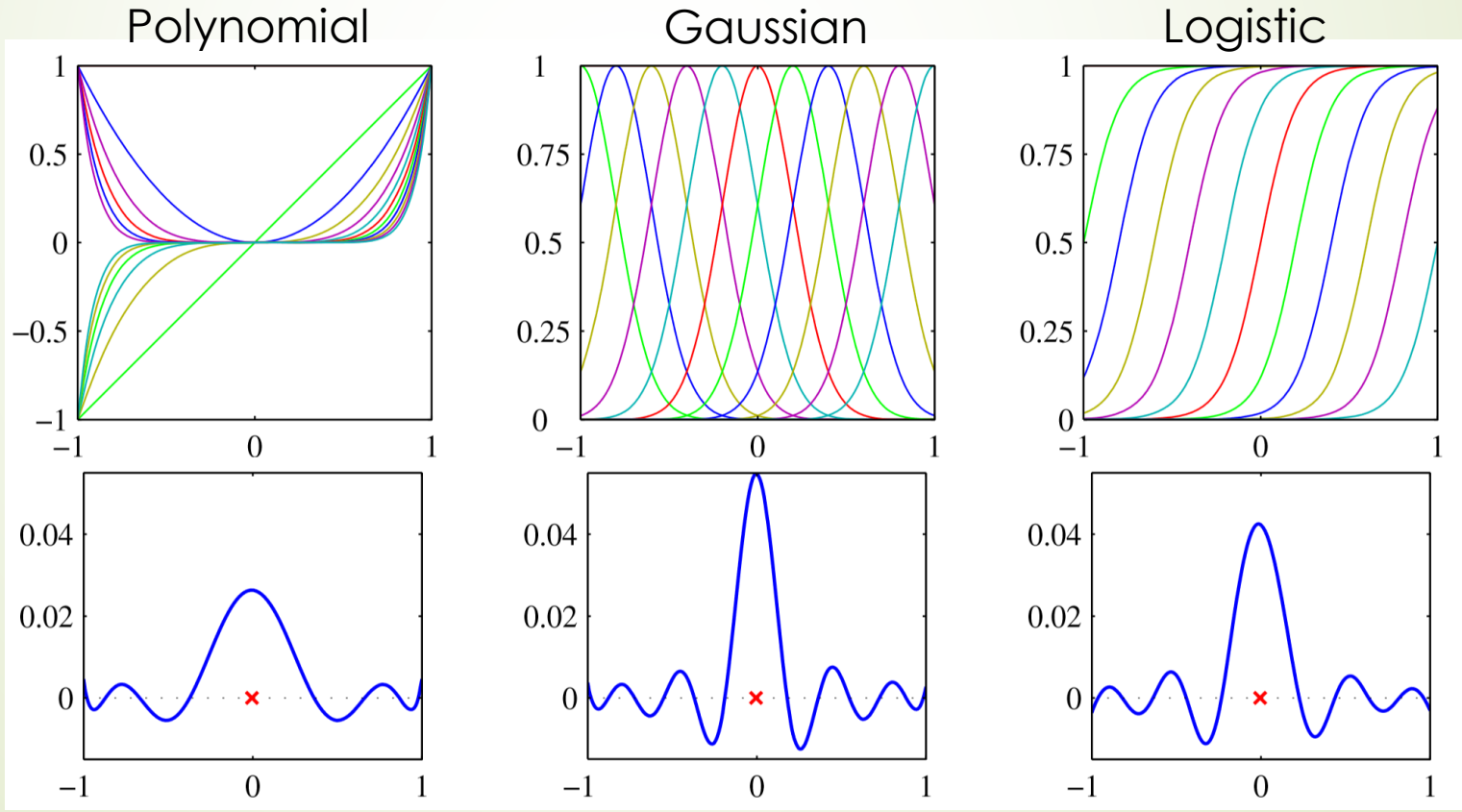
$$\mathbf{k}(\mathbf{x}) = \big(k(\mathbf{x}_1, \mathbf{x}), \cdots, k(\mathbf{x}_N, \mathbf{x})\big)$$

- Inverting $N \times N$ matrix instead of $M \times M$

Constructing Kernels - First Approach

- Choose feature space mapping $\boldsymbol{\phi}(\mathbf{x})$

$$k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{x}') = \sum_{i=1}^{M} \phi_i(\mathbf{x}) \phi_i(\mathbf{x}')$$



Constructing Kernels - Second Approach

- Construct kernel function directly and verify its validity
- Simple example

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

in 2-D case corresponds to

$$k(\mathbf{x}, \mathbf{z}) = \boldsymbol{\phi}(\mathbf{x})^\mathsf{T}\boldsymbol{\phi}(\mathbf{z})$$

with $\boldsymbol{\phi}(\mathbf{x}) = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right)^\mathsf{T}$

To test validity without having to construct the function $\boldsymbol{\phi}(\mathbf{x})$ explicitly, one can use the condition:

Function $k(\mathbf{x}, \mathbf{x}')$ is a valid kernel $\Leftrightarrow \mathbf{K} \geq 0 \quad \forall\{x_n\}$

Combining Kernels

- Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$ the following kernels will also be valid:

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= c k_1(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= q\left(k_1(\mathbf{x}, \mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}' \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) k_b(\mathbf{x}_b, \mathbf{x}'_b)
\end{aligned}
$$

with corresponding conditions on $c, f, q, \phi, k_3, \mathbf{A}, \mathbf{x}_a, \mathbf{x}_b, k_a, k_b$

Examples

- Polynomial kernels:
  $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\mathsf{T}\mathbf{x}')^2$ contains only terms of degree 2
  $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\mathsf{T}\mathbf{x}' + c)^2,\ c > 0$
  $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\mathsf{T}\mathbf{x}')^M$ contains only terms of degree M
  $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\mathsf{T}\mathbf{x}' + c)^M,\ c > 0$
- Gaussian kernel:
  $k(\mathbf{x}, \mathbf{x}') = \exp\left(-\|\mathbf{x} - \mathbf{x}'\|^2 / 2\sigma^2\right)$
- Note: can substitute $\mathbf{x}^\mathsf{T}\mathbf{x}'$ with a nonlinear kernel $\kappa(\mathbf{x}, \mathbf{x}')$
- Kernel on nonvectorial space:
  $k(A_1, A_2) = 2^{|A_1 \cap A_2|}$
- Sigmoid kernel:
  $k(\mathbf{x}, \mathbf{x}') = \tanh(a\mathbf{x}^\mathsf{T}\mathbf{x}' + b)$

Probabilistic Generative Models

- Kernel for generative model $p(\mathbf{x})$:

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

$$k(\mathbf{x}, \mathbf{x}') = \sum_i p(\mathbf{x}|i)p(\mathbf{x}'|i)p(i)$$

$$k(\mathbf{x}, \mathbf{x}') = \int p(\mathbf{x}|z)p(\mathbf{x}'|z)p(z)dz$$

- Kernel for HMM:

$$k(\mathbf{X}, \mathbf{X}') = \sum_{\mathbf{Z}} p(\mathbf{X}|\mathbf{Z})p(\mathbf{X}'|\mathbf{Z})p(\mathbf{Z})$$

- $\mathbf{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_L\}$ – observations
- $\mathbf{Z} = \{\mathbf{z}_1, \cdots, \mathbf{z}_L\}$ – hidden states

Fisher Kernel

- Parametric generative model $p(\mathbf{x}|\boldsymbol{\theta})$
- Fisher score:

$$\mathbf{g}(\boldsymbol{\theta}, \mathbf{x}) = \nabla_{\boldsymbol{\theta}} \ln p(\mathbf{x}|\boldsymbol{\theta})$$

- Fisher kernel and information matrix:

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{g}(\boldsymbol{\theta}, \mathbf{x})^{\mathsf{T}} \mathbf{F}^{-1} \mathbf{g}(\boldsymbol{\theta}, \mathbf{x}')$$

$$\mathbf{F} = \mathbb{E}_{\mathbf{x}}[\mathbf{g}(\boldsymbol{\theta}, \mathbf{x}) \mathbf{g}(\boldsymbol{\theta}, \mathbf{x})^{\mathsf{T}}]$$

- Note: the kernel is invariant under $\boldsymbol{\theta} \rightarrow \boldsymbol{\psi}(\boldsymbol{\theta})$
- Simplify matrix calculation:

$$\mathbf{F} \simeq \frac{1}{N} \sum_{n=1}^{N} \mathbf{g}(\boldsymbol{\theta}, \mathbf{x}) \mathbf{g}(\boldsymbol{\theta}, \mathbf{x})^{\mathsf{T}}$$

- Or simply omit the Fisher information matrix

Radial Basis Functions

- Consider basis function depends only on the radial distance (typically Euclidean) from a center $\boldsymbol{\mu}_j$

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

- Originally were introduced for the problem of exact function interpolation $f(\mathbf{x}_n) = t_n$:

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

- Interpolation problem with noisy inputs $\mathbf{x}_n$

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

$\nu(\boldsymbol{\xi})$: input noise distribution

Interpolation With Noisy Inputs

- Sum of squares function

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

- Optimal value

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

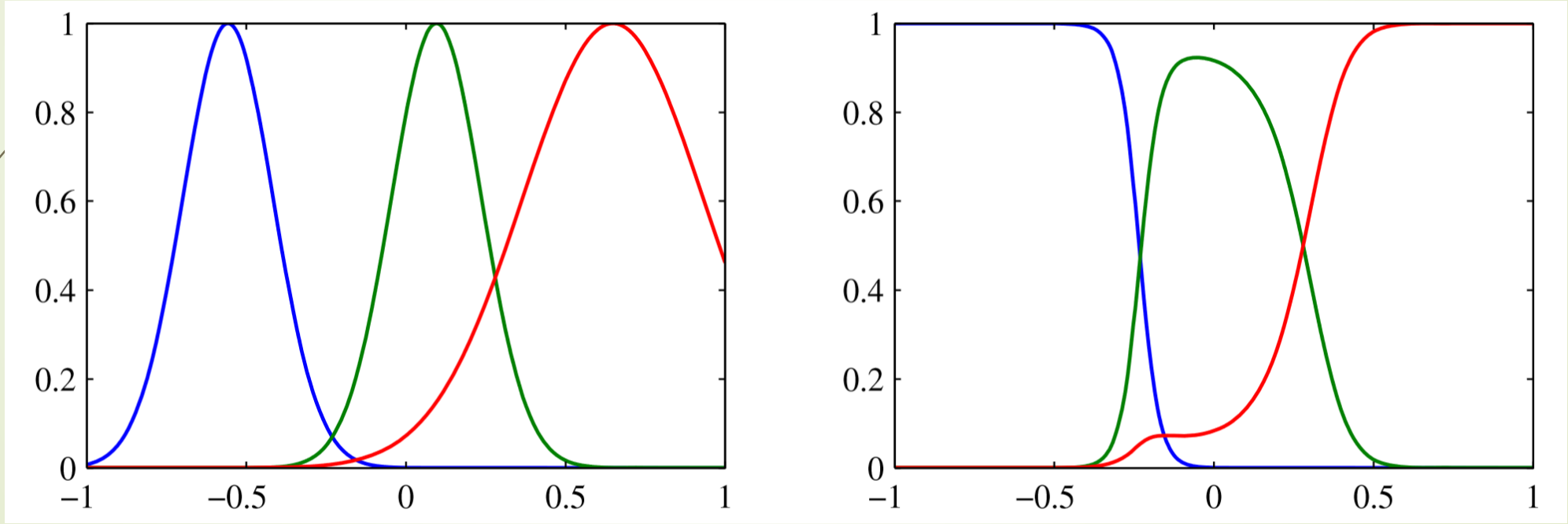
with basis functions given by normalized functions

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

- If noise distribution $\nu(\boldsymbol{\xi})$ is isotropic, basis functions would be radial

Normalization effect

- Avoids regions in an input space where all of the basis functions take small values



Reducing Size of the Basis

- Keep number of basis functions M smaller than input data size N
- Center locations $\boldsymbol{\mu}_i$ are determined based on the input data $\{\mathbf{x}_n\}$ alone
- Coefficients $\{w_i\}$ are determined by least squares
- Choice of centers:
  - Random
  - orthogonal least squares - greatest error reduction
  - using clustering algorithms

Parzen Density Estimator

- Prediction of linear regression model - linear combination of t_n with "equivalent kernel" values
- Same result starting from Parzen density estimator

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

- Regression function

$$y(\mathbf{x}) = \mathbb{E}[t|\mathbf{x}] = \int_{-\infty}^{\infty} tp(t|\mathbf{x})dt = \frac{\int tp(\mathbf{x}, t)dt}{\int p(\mathbf{x}, t)dt}$$

$$= \frac{\sum_n \int tf(\mathbf{x} - \mathbf{x}_n, t - t_n)dt}{\sum_m \int f(\mathbf{x} - \mathbf{x}_m, t - t_m)dt}$$

Nadaraya-Watson Model

Assume that the component density functions have zero mean so that

$$\int_{-\infty}^{\infty} t f(\mathbf{x}, t) dt = 0$$

for all values of $\mathbf{x}$. Then by variable change

$$y(\mathbf{x}) = \frac{\sum_n g(\mathbf{x} - \mathbf{x}_n) t_n}{\sum_m g(\mathbf{x} - \mathbf{x}_m)} = \sum_n k(\mathbf{x}, \mathbf{x}_n) t_n$$

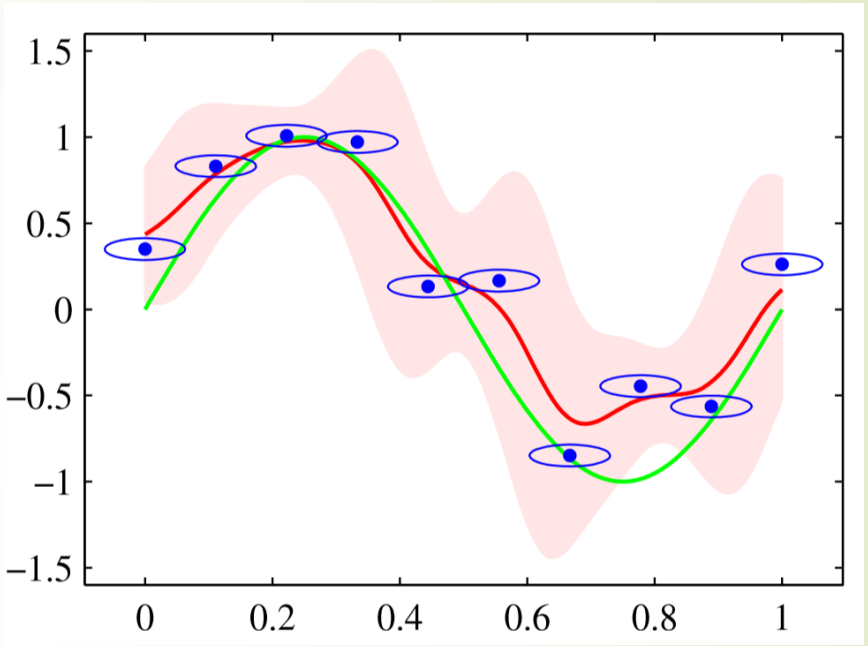
(Nadaraya-Watson Model)

with $g(\mathbf{x}) = \int_{-\infty}^{\infty} f(\mathbf{x}, t) dt$ and $k(\mathbf{x}, \mathbf{x}_n)$ given by

$$k(\mathbf{x}, \mathbf{x}_n) = \frac{g(\mathbf{x} - \mathbf{x}_n)}{\sum_m g(\mathbf{x} - \mathbf{x}_m)}$$

Illustration of the Nadaraya-Watson Model

single input variable x in which $f(x,t)$ is a zero-mean isotropic Gaussian with variance σ^2



Conditional distribution

$$p(t|\mathbf{x}) = \frac{p(\mathbf{x},t)}{\int p(\mathbf{x},t)dt} = \frac{\sum_n f(\mathbf{x}-\mathbf{x}_n, t-t_n)}{\sum_m \int f(\mathbf{x}-\mathbf{x}_m, t-t_m)dt}$$

is given by a mixture of Gaussians

Gaussian Processes: Key Idea

- The idea is similar to linear regression with a fixed set of basis functions (Chapter 3):

$$y(\mathbf{x}) = \mathbf{w}^{\mathsf{T}}\boldsymbol{\phi}(\mathbf{x})$$

- However, we forget about the parametric model
- Instead we take an infinite number of basis functions given by a probability distribution over functions
- Might look difficult to handle, but it is not… we only have to consider the values of the functions at training/test data points

Linear Regression Revisited

- Model defined by linear combination of M fixed basis functions:

$$y(\mathbf{x}) = \mathbf{w}^{\mathsf{T}}\boldsymbol{\phi}(\mathbf{x})$$

- A Gaussian prior distribution over the weight vector $\mathbf{w}$

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})$$

governed by the hyperparameter α (precision) of the distribution.

- $p(\mathbf{w})$ induces a probability distribution over functions $y(\mathbf{x})$.

Linear Regression Revisited

- Evaluating $y(\mathbf{x})$ for a set of training data points $\mathbf{x}_1, \ldots, \mathbf{x}_N$, we have a joint distribution $\mathbf{y} = \boldsymbol{\Phi}\mathbf{w}$, with elements $y_n = y(\mathbf{x}_n) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}_n)$, where $\boldsymbol{\Phi}$ is the design matrix with elements $\Phi_{nk} = \phi_k(\mathbf{x}_n)$.
- Since $\mathbf{y} = \boldsymbol{\Phi}\mathbf{w}$ is a linear combination of Gaussina distributed variables $\mathbf{w}$, $p(\mathbf{y})$ is also Gaussian, with mean and covariance:

$$\mathbb{E}[\mathbf{y}] = \mathbf{0}$$

$$\mathrm{Cov}[\mathbf{y}] = \frac{1}{\alpha}\boldsymbol{\Phi}\boldsymbol{\Phi}^\top = \mathbf{K}$$

where $\mathbf{K}$ is the Gram matrix with elements

$$K_{mn} = k(\mathbf{x}_m, \mathbf{x}_n) = \frac{1}{\alpha}\boldsymbol{\phi}(\mathbf{x}_n)^\top \boldsymbol{\phi}(\mathbf{x}_m)$$

and $k(\mathbf{x}, \mathbf{x}')$ is the kernel function

- Up to now we only took data points + prior, but no target values

Linear Regression Revisited: Summary

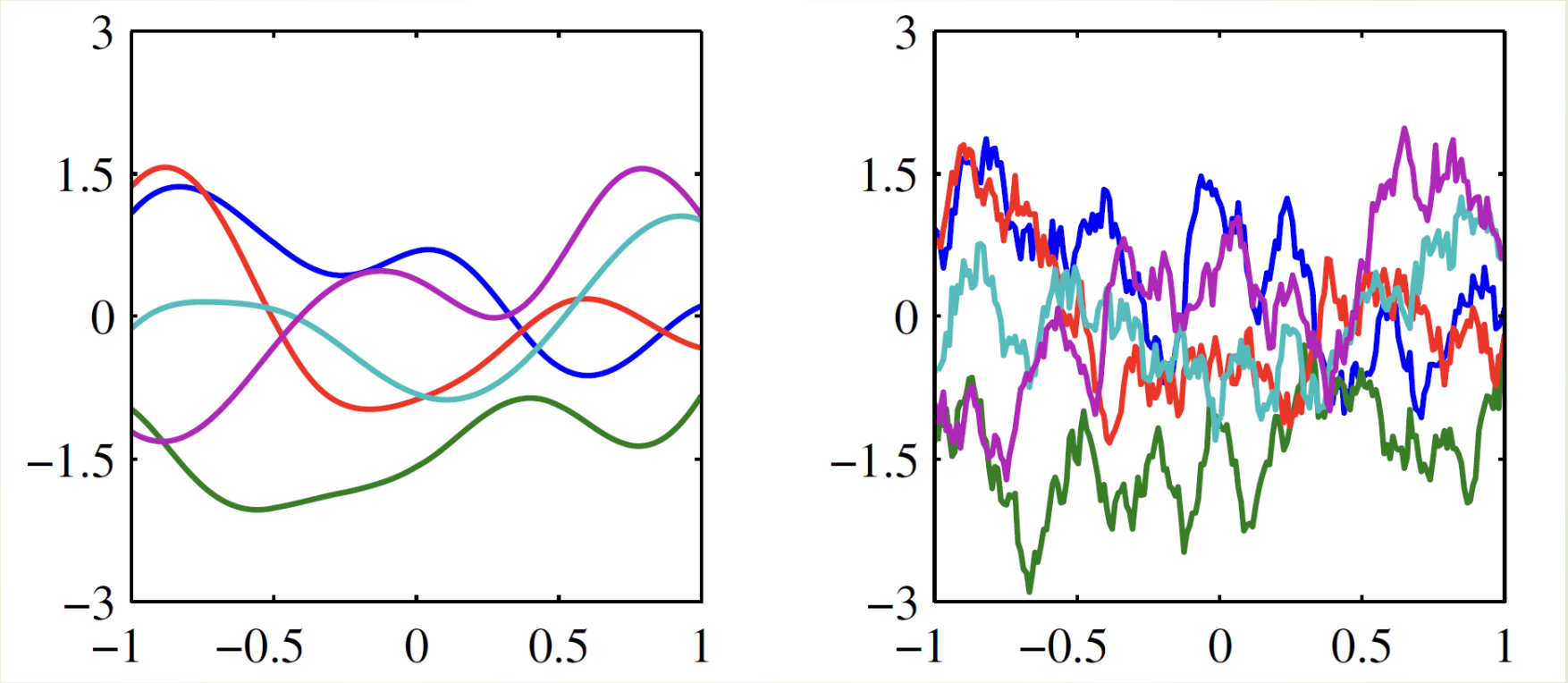
- The model we have seen is one example for a Gaussian process
- "Gaussian process is defined as a probability distribution over functions $y(\mathbf{x})$ such that the set of values of $y(\mathbf{x})$ evaluated at an arbitrary set of points $\mathbf{x}_1, \cdots, \mathbf{x}_N$ jointly have a Gaussian distribution"
- Key point: the joint distribution is specified completely by second-order statistics (mean, covariance)
- Note, usually the mean is taken to be zero, then we only need the covariance, i.e., the kernel-function:

$$\mathbb{E}[y(\mathbf{x}_n)y(\mathbf{x}_m)] = k(\mathbf{x}_n, \mathbf{x}_m)$$

- Actually, instead of choosing (a limited number of) basis functions, we can directly choose a kernel function (which may result in an infinite number of basis functions)

Examples of Gaussian Processes

Samples from Gaussian processes for a 'Gaussian' kernel (left) and an exponential kernel (right).



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

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

Gaussian Process Regression (1/3)

- To use Gaussian processes for regression, we need to model noise

$$t_n = y_n + \varepsilon_n \text{ with } y_n = y(\mathbf{x}_n)$$

- For noise processing with a Gaussian distribution we obtain

$$p(t_n|y_n) = \mathcal{N}(t_n|y_n, \beta^{-1})$$

- The joint distribution for $\mathbf{t} = (t_1, \ldots, t_N)^{\mathrm{T}}$ and $\mathbf{y} = (y_1, \ldots, y_N)^{\mathrm{T}}$ is then

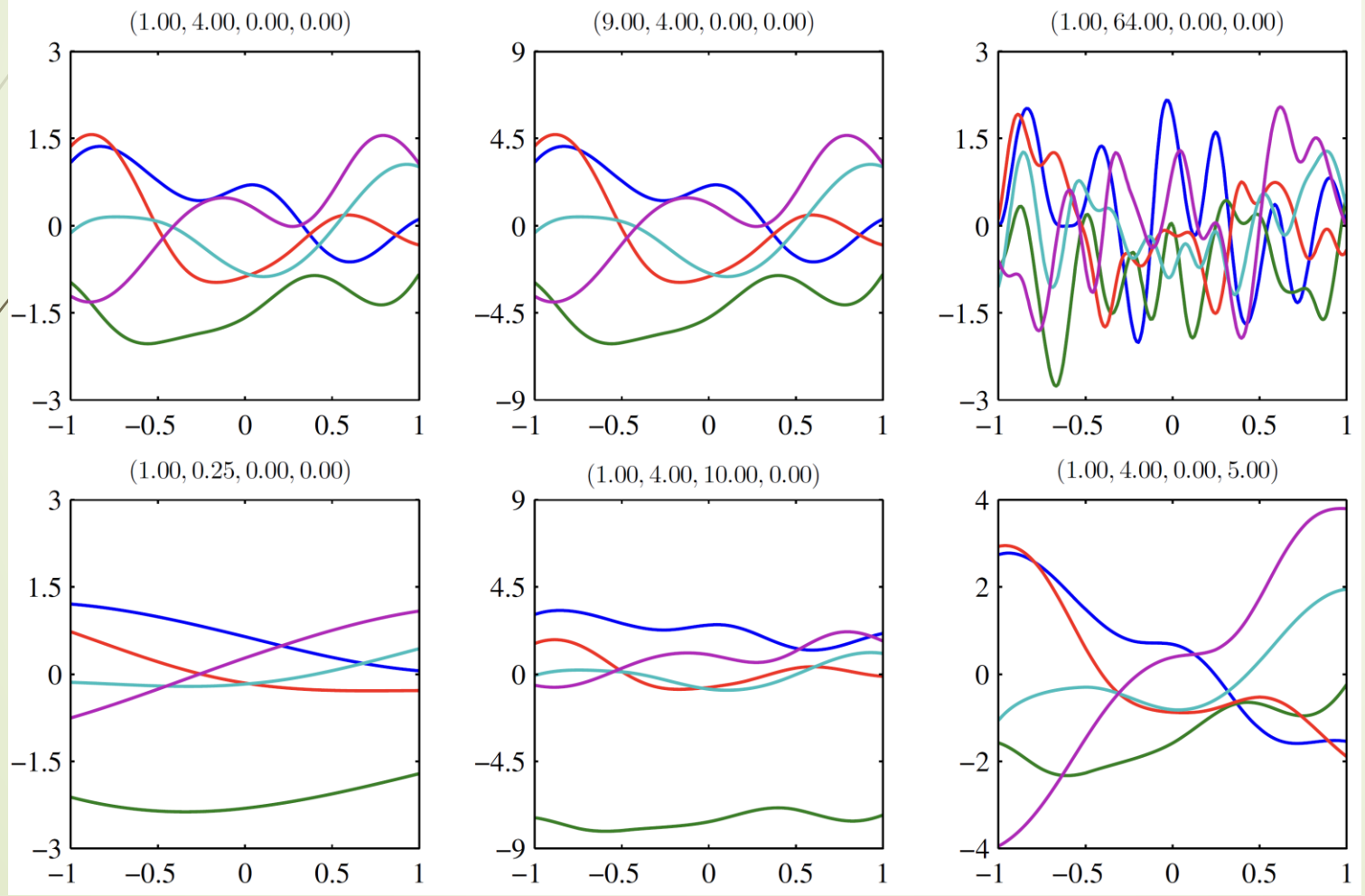
$$p(\mathbf{t}|\mathbf{y}) = \mathcal{N}(\mathbf{t}|\mathbf{y}, \beta^{-1}\mathbf{I}_N)$$

- From the definition of a Gaussian process we have $p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K})$
- The kernel function k is chosen to express the property that points that are more similar (given) will be stronger correlated.
- For the marginal distribution $p(\mathbf{t})$, we need to integrate over $\mathbf{y}$

$$p(\mathbf{t}) = \int p(\mathbf{t}|\mathbf{y})p(\mathbf{y})d\mathbf{y} = \mathcal{N}(\mathbf{t}|\mathbf{0}, \mathbf{C}) \text{ with } \mathbf{C} = \mathbf{K} + \beta^{-1}\mathbf{I}$$

Gaussian Process Regression (2/3)

$$k(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp\left\{-\frac{\theta_1}{2}\|\mathbf{x}_n - \mathbf{x}_m\|^2\right\} + \theta_2 + \theta_3 \mathbf{x}_n^{\mathsf{T}} \mathbf{x}_m$$



Samples from a Gaussian process prior defined by the covariance function. The title above each plot denotes $(\theta_0, \theta_1, \theta_2, \theta_3)$.

Gaussian Process Regression (3/3)

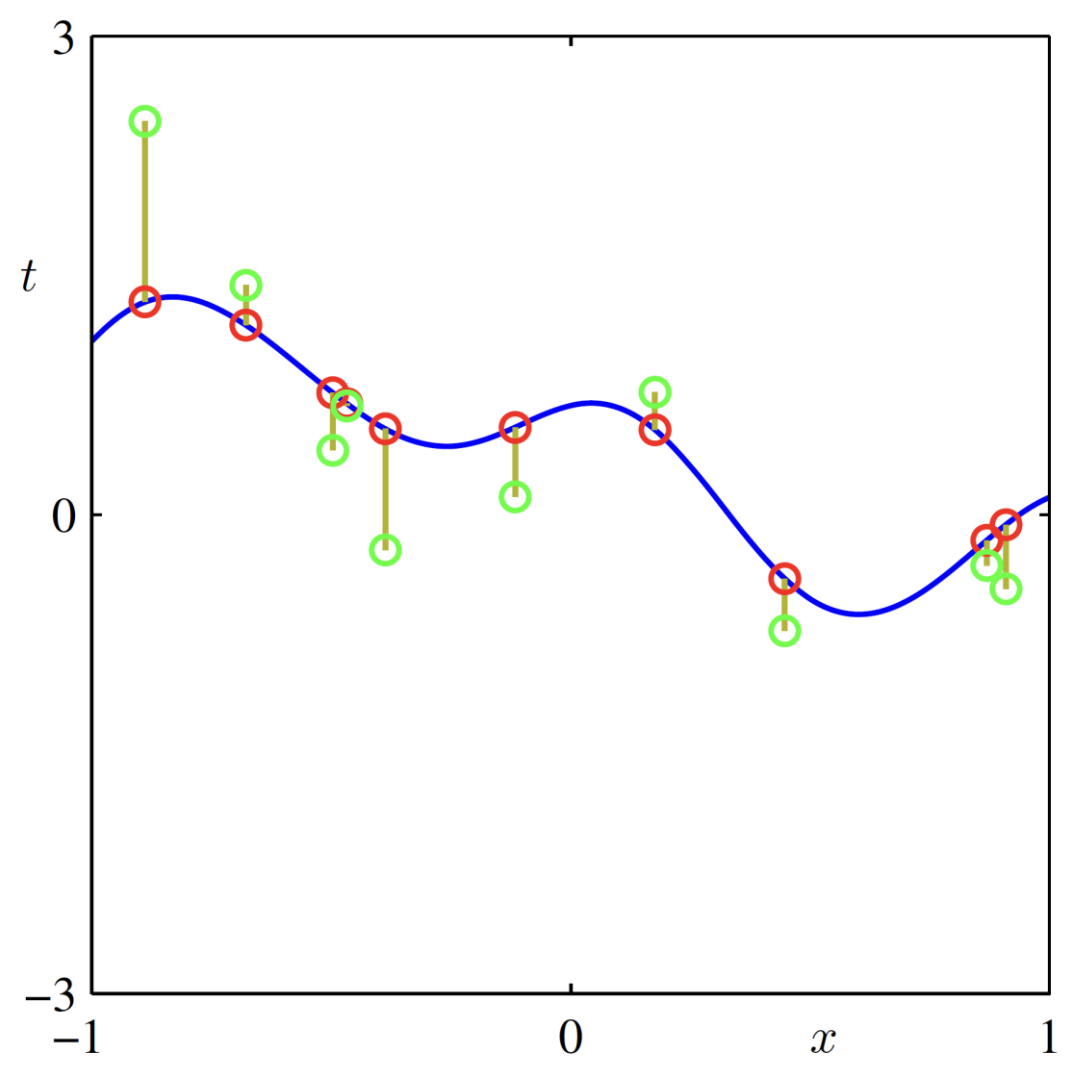


Illustration of the sampling of data points $\{t_n\}$ from a Gaussian process. The blue curve shows a sample function from the Gaussian process prior over functions, and the red points show the values of y_n obtained by evaluating the function at a set of input values $\{x_n\}$. The corresponding values of $\{t_n\}$, shown in green, are obtained by adding independent Gaussian noise to each of the $\{y_n\}$.

Making Predictions

- So far we have a model for the joint probability distribution over sets of data points
- For predictions of a new input variable $\mathbf{x}_{N+1}$, we need to evaluate the predictive distribution $p(t_{N+1}|\mathbf{t})$
- By partitioning (see Section 2.3.1) the joint Gaussian distribution over $\mathbf{x}_1, \cdots, \mathbf{x}_N, \mathbf{x}_{N+1}$, we obtain $p(t_{N+1}|\mathbf{t})$ given by its mean and covariance:

$$m(\mathbf{x}_{N+1}) = \mathbf{k}^{\mathsf{T}}\mathbf{C}_N^{-1}\mathbf{t}$$

$$\sigma^2(\mathbf{x}_{N+1}) = c - \mathbf{k}^{\mathsf{T}}\mathbf{C}_N^{-1}\mathbf{k}$$

$$\mathbf{k} = \left(k(\mathbf{x}_1, \mathbf{x}_{N+1}), \cdots, k(\mathbf{x}_N, \mathbf{x}_{N+1})\right)^{\mathsf{T}}$$

$$c = k(\mathbf{x}_{N+1}, \mathbf{x}_{N+1}) + \beta^{-1}$$

Making Predictions: $p(t_{N+1}|\mathbf{t})$

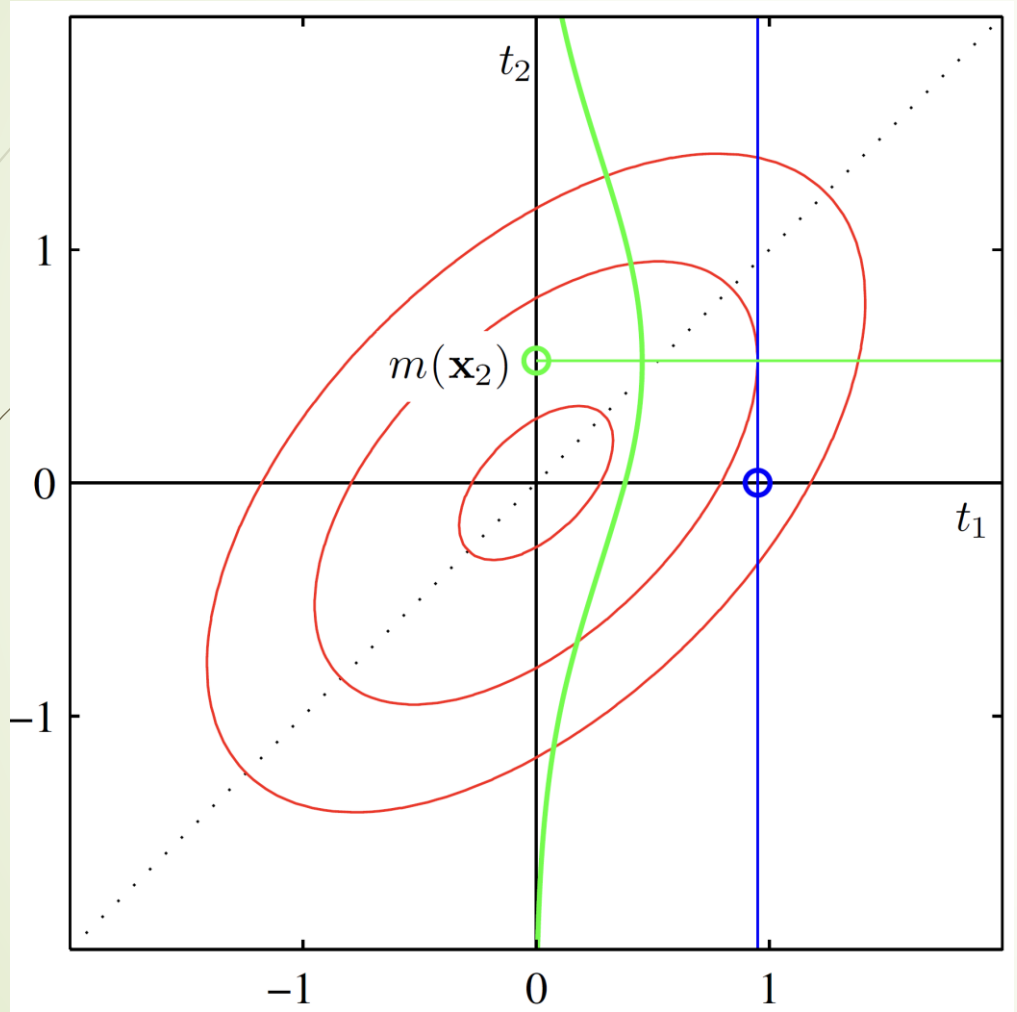
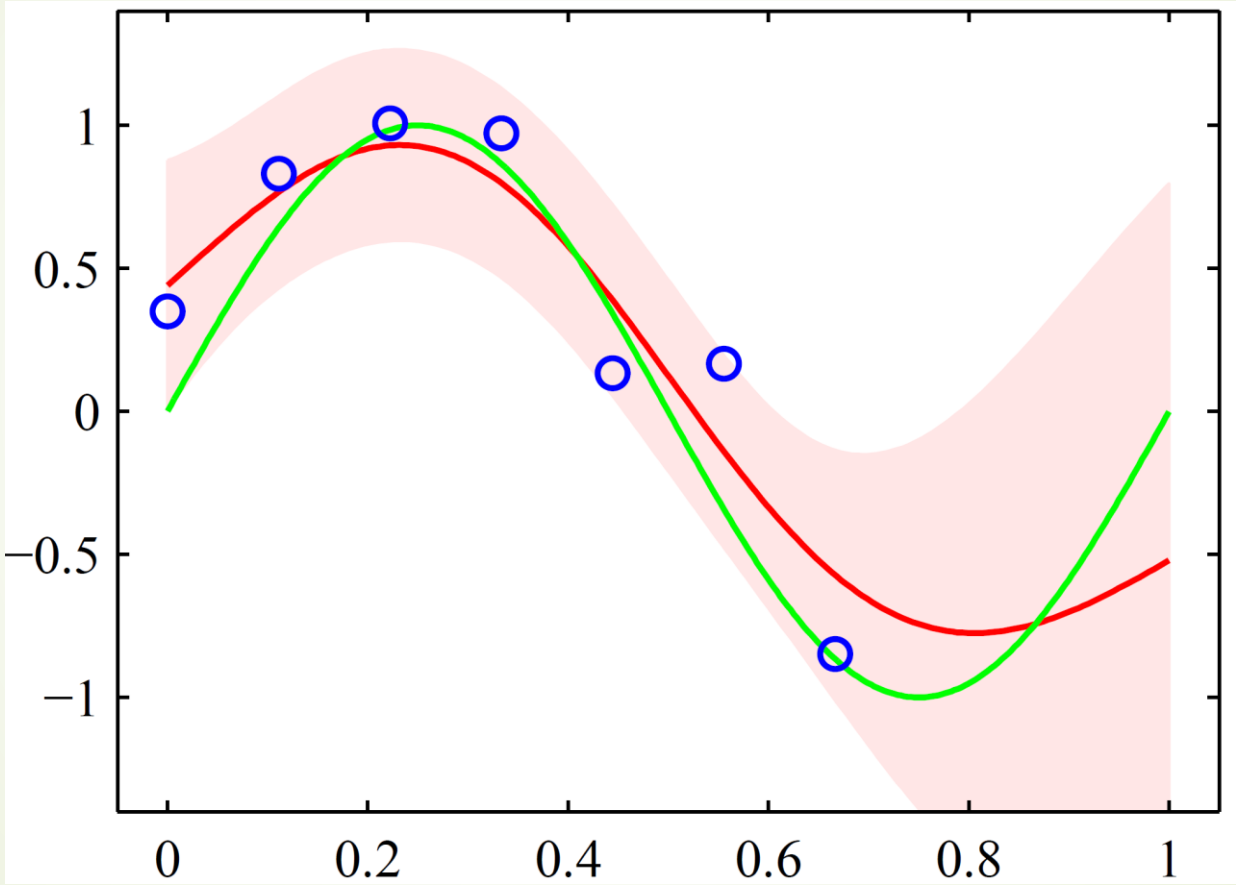


Illustration of the mechanism of Gaussian process regression for the case of one training point and one test point, in which the red ellipses show contours of the joint distribution $p(t_1, t_2)$. Here t_1 is the training data point, and conditioning on the value of t_1, corresponding to the vertical blue line, we obtain $p(t_2|t_1)$ shown as a function of t_2 by the green curve.

Example of Gaussian Process Regression



Green curve: original sinusoidal function; blue points: sampled training data points with additional noise; read line: mean estimate; shaded regions: $\pm 2\sigma$

Learning Hyperparameters $\boldsymbol{\theta}$

- In practice, it can be preferable not to fix parameters, but to infer them from the data
- Parameters $\boldsymbol{\theta}$ are, e.g.: length scale of correlations, precision of noise (β)
- Simplest approach:
  - Maximizing the log-likelihood $\ln p(\mathbf{t}|\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$ (maximum likelihood)
  - Problem: $\ln p(\mathbf{t}|\boldsymbol{\theta})$ is in general non-convex and can have multiple maxima
- Introduce a prior $p(\boldsymbol{\theta})$ and maximize the log-posterior (maximum a posteriori): $\ln p(\mathbf{t}|\boldsymbol{\theta}) + \ln p(\boldsymbol{\theta})$
- To be Bayesian, we need the actual distribution and have to marginalize; this is not tractable $\Rightarrow$ approximations
- The noise might not be additive but dependent on $\mathbf{x}$
  - A second Gaussian process can be introduced to represent the dependency of β on $\mathbf{x}$

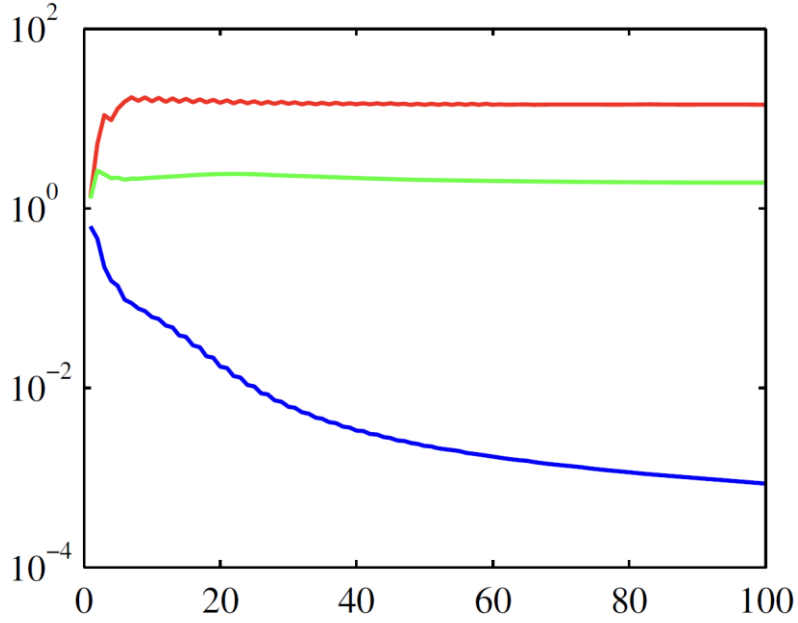
Automatic Relevance Detection (ARD)

- In practice, it is preferable to infer parameters from the data

$$k(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp\left\{-\frac{1}{2}\sum_{i=1}^{D} \eta_i (x_{ni} - x_{mi})^2\right\} + \theta_2 + \theta_3 \mathbf{x}_n^{\mathrm{T}} \mathbf{x}_m$$

- Hyperparameter optimization by maximum likelihood allows a different weighting of each dimension
- Irrelevant dimensions (with small weights) can be detected and discarded

Illustration of automatic relevance determination in a Gaussian process for a synthetic problem having three inputs x_1, x_2, and x_3, for which the curves show the corresponding values of the hyperparameters η_1 (red), η_2 (green), and η_3 (blue) as a function of the number of iterations when optimizing the marginal likelihood. Details are given in the text. Note the logarithmic scale on the vertical axis.



Gaussian Process for Classification (1/3)

- **Objective:** model posterior probabilities of the target variable
- **Problem:** we need to map values to interval (0, 1)
- **Solution:** use a Gaussian process together with a non-linear activation function (e.g., sigmoid)

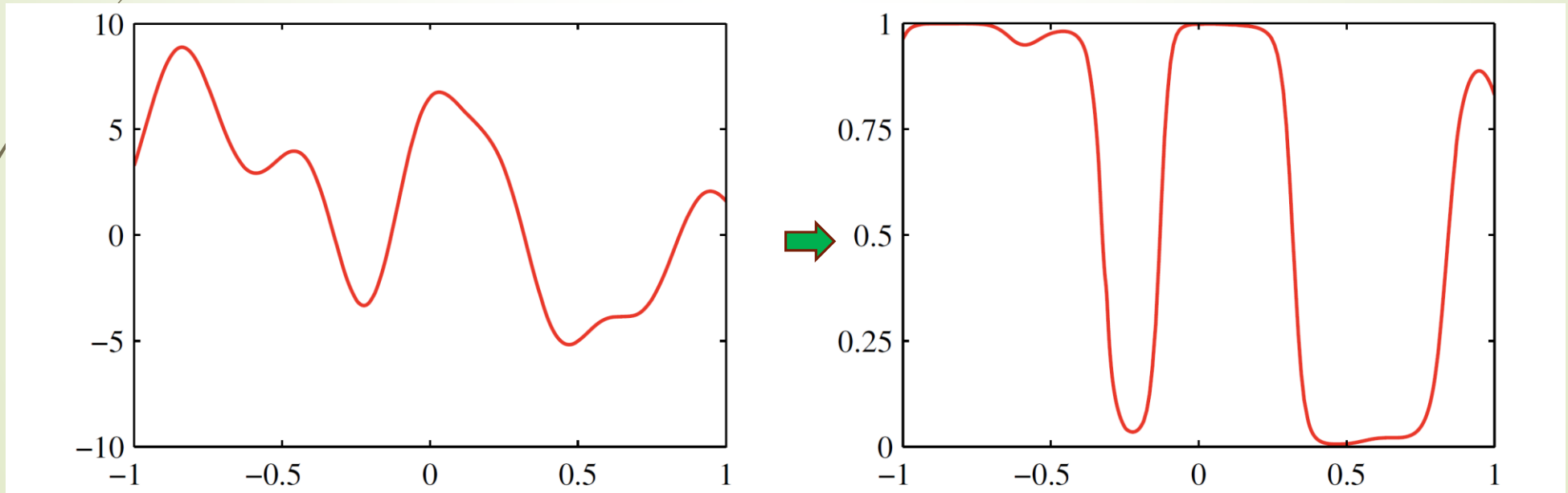


Figure 6.11 The left plot shows a sample from a Gaussian process prior over functions $a(\mathbf{x})$, and the right plot shows the result of transforming this sample using a logistic sigmoid function.

Gaussian Process for Classification (2/3)

- Consider two-class problem with target values $t \in \{0, 1\}$. Define a Gaussian process over a function $a(\mathbf{x})$ and transform a using the logistic sigmoid to

$$y = \sigma\big(a(\mathbf{x})\big)$$

- Similar to before, we need to predict the conditional distribution:

$$p(t_{N+1} = 1|\mathbf{t}) = \int p(t_{N+1} = 1|a_{N+1})p(a_{N+1}|\mathbf{t})da_{N+1}$$

$$= \int \sigma(a_{N+1})p(a_{N+1}|\mathbf{t})da_{N+1}$$

- However, the integral is analytically intractable. Approximations can be of numerical or analytical nature.

Gaussian Process for Classification (3/3)

- Problem 1: we need to compute a weird integral
- Solution 1: we know how to compute the convolution of an Gaussian and a sigmoid function (Eq. (4.153)) approximate the posterior distribution $p(a_{N+1}|\mathbf{t})$ as Gaussian
- Problem 2: But how do we approximate the posterior?
- Solution 2: the Laplace approximation (among others)

Laplace Approximation (1/2)

- We can rewrite the posterior over a_{N+1} using Bayes' theorem:

$$p(a_{N+1}|\mathbf{t}) = \int p(a_{N+1}, \mathbf{a}|\mathbf{t})d\mathbf{a} = \cdots = \int p(a_{N+1}|\mathbf{a})p(\mathbf{a}|\mathbf{t})d\mathbf{a}$$

- We know how to compute the mean and covariance for $p(a_{N+1}|\mathbf{a})$.

$$p(a_{N+1}|\mathbf{a}) = \mathcal{N}\left(a_{N+1}\middle|\mathbf{k}^{\mathrm{T}}\mathbf{C}_N^{-1}\mathbf{a}, c - \mathbf{k}^{\mathrm{T}}\mathbf{C}_N^{-1}\mathbf{k}\right)$$

- Now we have to find a Gaussian approximation only for $p(\mathbf{a}|\mathbf{t})$. This is done noting that $p(\mathbf{a}|\mathbf{t}) \propto p(\mathbf{t}|\mathbf{a})p(\mathbf{a})$, thus

$$\Psi(\mathbf{a}) = \ln p(\mathbf{a}|\mathbf{t}) = \ln p(\mathbf{t}|\mathbf{a}) + \ln p(\mathbf{a}) + \text{const}$$

Laplace Approximation (2/2)

- We can use the iterative reweighted least squares (IRLS) algorithm (Sec. 4.3.3) to find the mode of $\Psi(\mathbf{a})$ (first and second derivative have to be evaluated)
- It can be shown that $\Psi(\mathbf{a})$ is convex and thus has only one mode
- The mode position $\mathbf{a}^*$ and the Hessian matrix $\mathbf{H}$ at this position define our Gaussian approximation

$$q(\mathbf{a}) = \mathcal{N}(\mathbf{a}|\mathrm{a}^*, \mathbf{H}^{-1})$$

- Now we can go back to the formulas and compute the integrals and finally also $p(a_{N+1}|\mathbf{t})$ from

$$p(t_{N+1}|\mathbf{t}) = \int p(t_{N+1}|a_{N+1})p(a_{N+1}|\mathbf{t})da_{N+1}$$

Learning Hyperparameters

- To determine the parameters , we can maximize the likelihood function:

$$p(\mathbf{t}|\boldsymbol{\theta}) = \int p(\mathbf{t}|\mathbf{a})p(\mathbf{a}|\boldsymbol{\theta})d\mathbf{a}$$

- Again, the integral is analytically intractable, so the Laplace approximation can be applied again
- We need an expression for the gradient of the logarithm of

Example Kernels

- Linear kernel

$$k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z}$$

- Gaussian (Radial basis function) kernel

$$k(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{z})^\top \Sigma^{-1}(\mathbf{x} - \mathbf{z})\right)$$

- Sigmoid kernel

$$k(\mathbf{x}, \mathbf{z}) = \tanh(a \cdot \mathbf{x}^\top \mathbf{z} + b)$$

Kernel

- $k(\cdot,\cdot)$ a legal definition of inner product:

$\exists\ \boldsymbol{\phi}: \mathbf{x} \rightarrow R^N$

s.t. $k(\mathbf{x}, \mathbf{z}) = \boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\phi}(\mathbf{z})$

Why Kernels Matter?

- Many algorithms interact with data only via **dot-products**
- Replace $\mathbf{x}^\mathsf{T}\mathbf{z}$ with $k(\mathbf{x}, \mathbf{z}) = \boldsymbol{\phi}(\mathbf{x})^\mathsf{T}\boldsymbol{\phi}(\mathbf{z})$
- Act **implicitly** as if data was in the higher-dimensional $\boldsymbol{\phi}$-space

Example

$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z})^2$ corresponds to

$$(x_1, x_2) \rightarrow \boldsymbol{\phi}(\mathbf{x}) = \left(x_1^2, x_2^2, \sqrt{2}x_1 x_2\right)$$

$$\boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\phi}(\mathbf{z}) = \left(x_1^2, x_2^2, \sqrt{2}x_1 x_2\right)\left(z_1^2, z_2^2, \sqrt{2}z_1 z_2\right)^\top$$

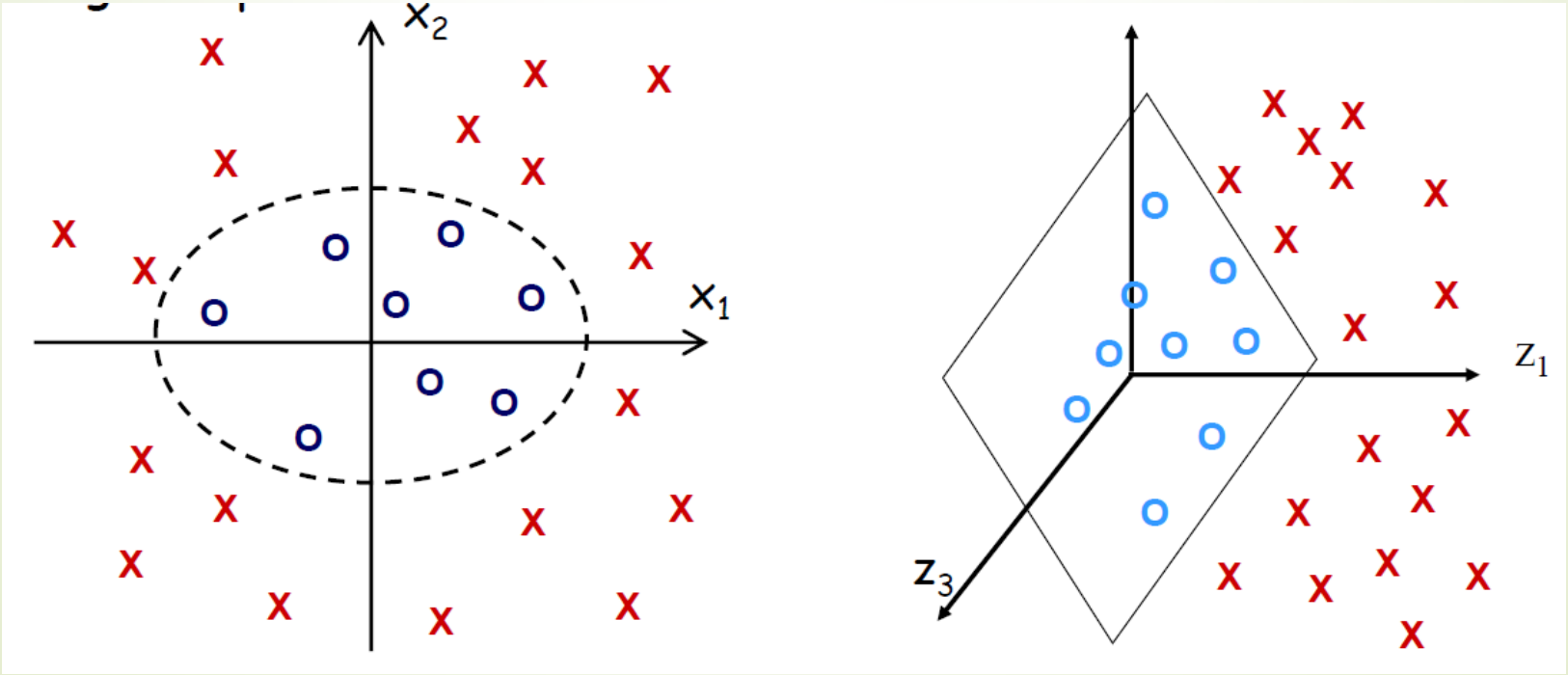
$$= x_1^2 z_1^2 + x_2^2 z_2^2 + 2\mathrm{x}_1\mathrm{x}_2\mathrm{z}_1\mathrm{z}_2 = (\mathrm{x}_1\mathrm{z}_1 + \mathrm{x}_2\mathrm{z}_2)^2$$

$$= (\mathbf{x}^\top \mathbf{z})^2 = k(\mathbf{x}, \mathbf{z})$$

Example

$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\mathsf{T}\mathbf{z})^2$ corresponds to

$(x_1, x_2) \rightarrow \boldsymbol{\phi}(\mathbf{x}) = \left(x_1^2, x_2^2, \sqrt{2}x_1x_2\right)$



Constructing New Kernels

- Positive scaling

$$k(\mathbf{x}, \mathbf{z}) = ck_1(\mathbf{x}, \mathbf{z}), c > 0$$

- Exponentiation

$$k(\mathbf{x}, \mathbf{z}) = \exp\big(k_1(\mathbf{x}, \mathbf{z})\big)$$

- Addition

$$k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z}) + k_2(\mathbf{x}, \mathbf{z})$$

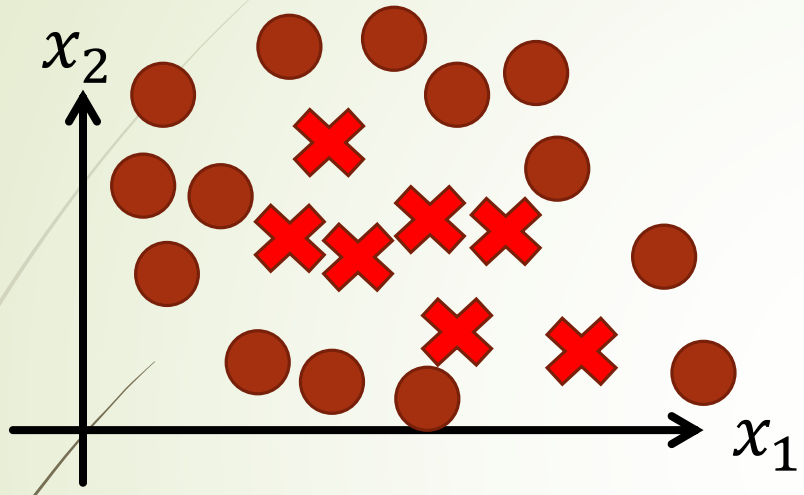
- Multiplication with function

$$k(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})k_1(\mathbf{x}, \mathbf{z})f(\mathbf{z})$$

- Multiplication

$$k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z})k_2(\mathbf{x}, \mathbf{z})$$

Non-linear decision boundary



Predict $y = 1$ if

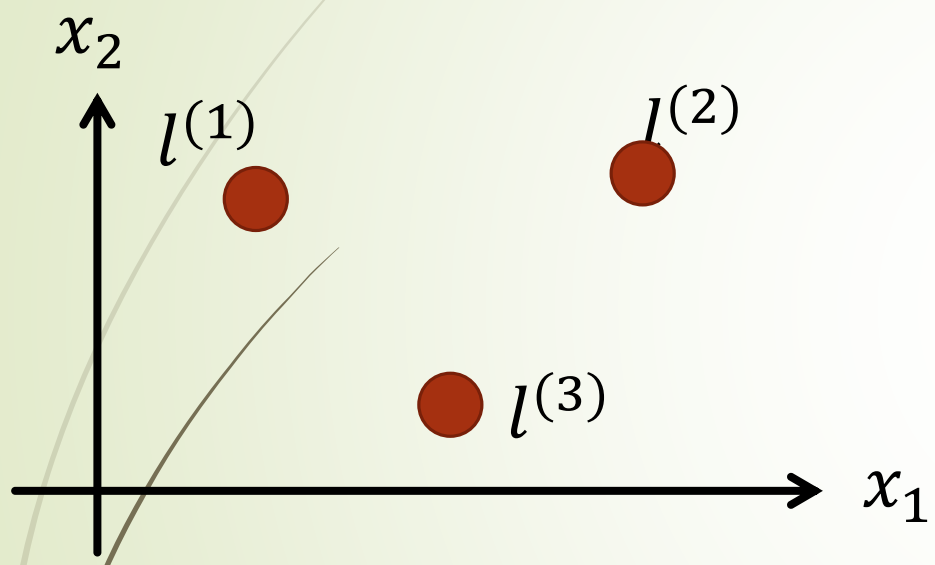
$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \cdots \geq 0$$

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \cdots$$

$$f_1 = x_1, f_2 = x_2, f_3 = x_1 x_2, \cdots$$

Is there a different/better choice of the features $f_1, f_2, f_3, \cdots$?

Kernel

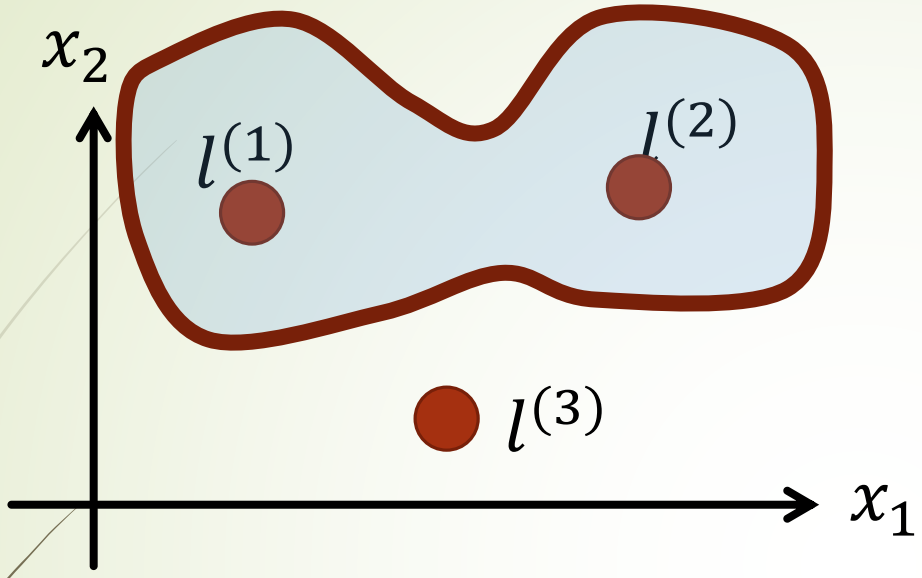


Give $\mathbf{x}$, compute new features depending on proximity to landmarks $l^{(1)}$, $l^{(2)}$, $l^{(3)}$

$$f_1 = \text{similarity}(\mathbf{x}, l^{(1)})$$
$$f_2 = \text{similarity}(\mathbf{x}, l^{(2)})$$
$$f_3 = \text{similarity}(\mathbf{x}, l^{(3)})$$

Gaussian kernel

$$\text{similarity}\left(x, l^{(i)}\right) = \exp\left(-\frac{\left\|x - l^{(i)}\right\|^2}{2\sigma^2}\right)$$



$$f_1 = \text{similarity}(x, l^{(1)})$$
$$f_2 = \text{similarity}(x, l^{(2)})$$
$$f_3 = \text{similarity}(x, l^{(3)})$$

Predict $y = 1$ if

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$$

Ex: $\theta_0 = -0.5, \theta_1 = 1, \theta_2 = 1, \theta_3 = 0$

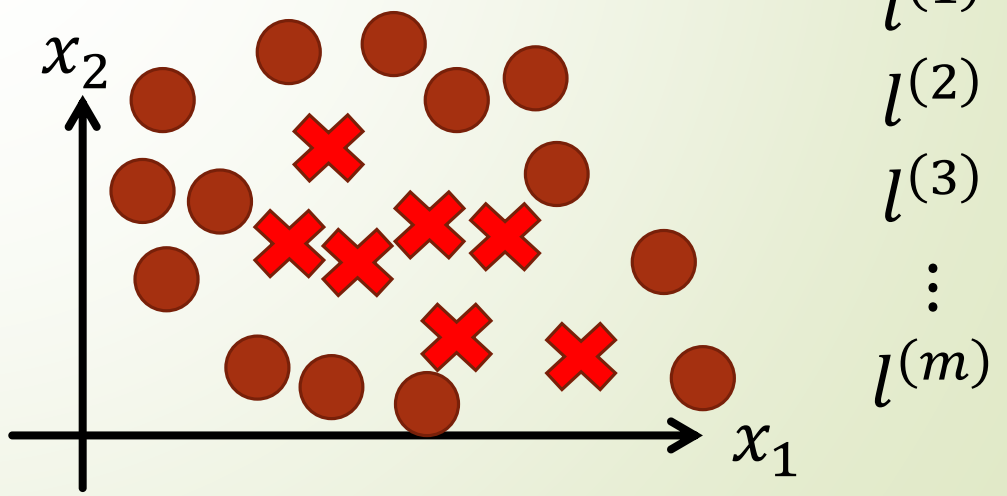
Choosing the landmarks

- Given x

$$f_i = \text{similarity}\left(x, l^{(i)}\right) = \exp\left(-\frac{\left\|x - l^{(i)}\right\|^2}{2\sigma^2}\right)$$

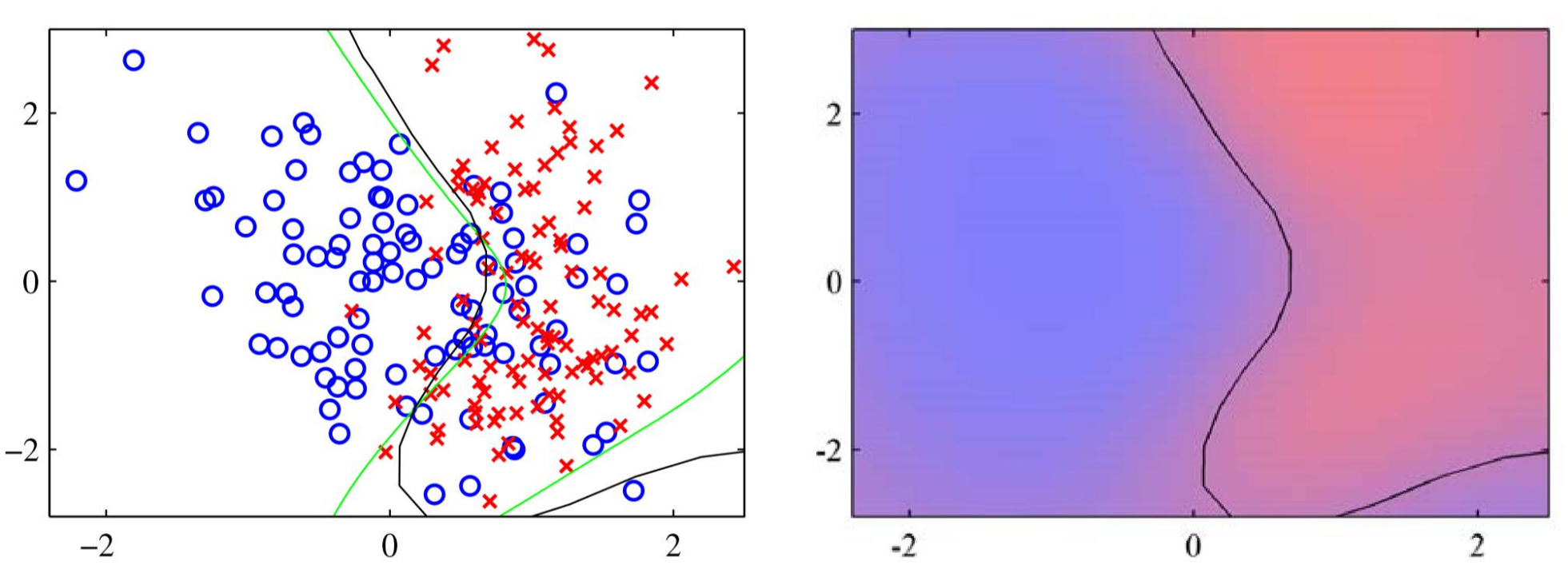
Predict $y = 1$ if $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

Where to get $l^{(1)}, l^{(2)}, l^{(3)}, \ldots$?



Slide credit: Andrew Ng

Gaussian Process for Classification



Connection to Neural Networks

- Neural Networks
  - The range of representable functions is governed by the number M of hidden units
  - Within the maximum likelihood framework, they overfit as M comes close to the number of training samples
- Bayesian Neural Networks
  - The prior over $\mathbf{w}$ in conjunction with the network function $f(\mathbf{x}, \mathbf{w})$ produces a prior distribution over functions from $y(\mathbf{x})$
  - The distribution of functions will tend to a Gaussian process in the limit $M \to \infty$
  - The property that the outputs share hidden units (and thus "borrow statistical strength" from each other) is lost in this limit