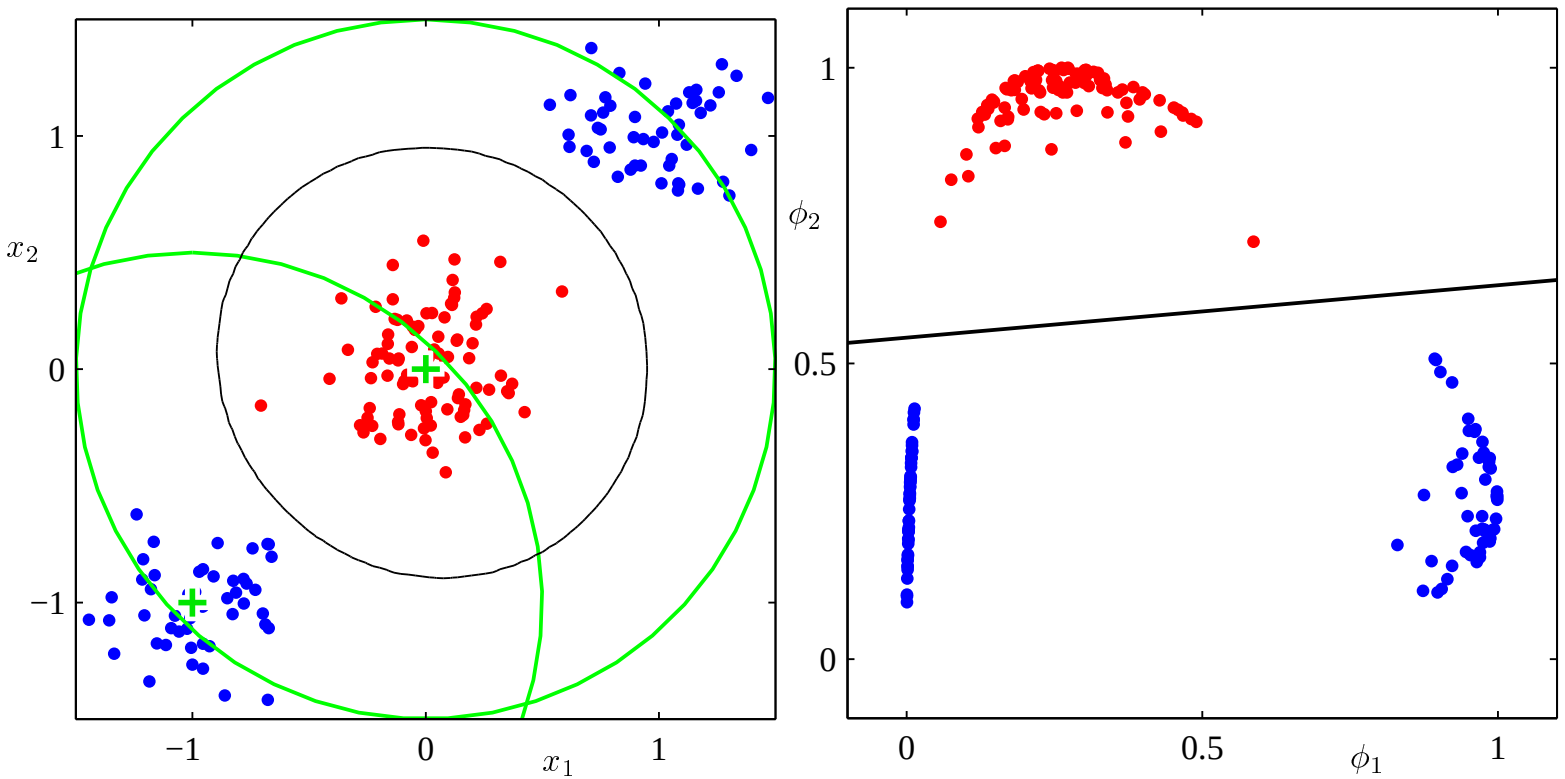
Lecture 9

Kernel Methods

University of Amsterdam

Basis Functions

Recall that by transforming the features, we could transform harder problems into easier ones



Parametric vs. Non-parametric

Last week, we saw one way to find good basis functions automatically:

- Multi-layer neural networks allow us to define functions over functions
- The clear distinction between feature extractor and classifier disappears
- Feature extractor adapts to the data
- Parametric method

Today we see another approach:

- Use data, not parameters, to describe the basis functions
- Similar to kNN
- Non-parametric method

Parametric vs. Non-parametric

Last week, we saw one way to find good basis functions automatically:

- Multi-layer neural networks allow us to define functions over functions
- The clear distinction between feature extractor and classifier disappears
- Feature extractor adapts to the data
- Parametric method

Today we see another approach:

- Use data, not parameters, to describe the basis functions
- Similar to kNN
- Non-parametric method

Kernel density estimation

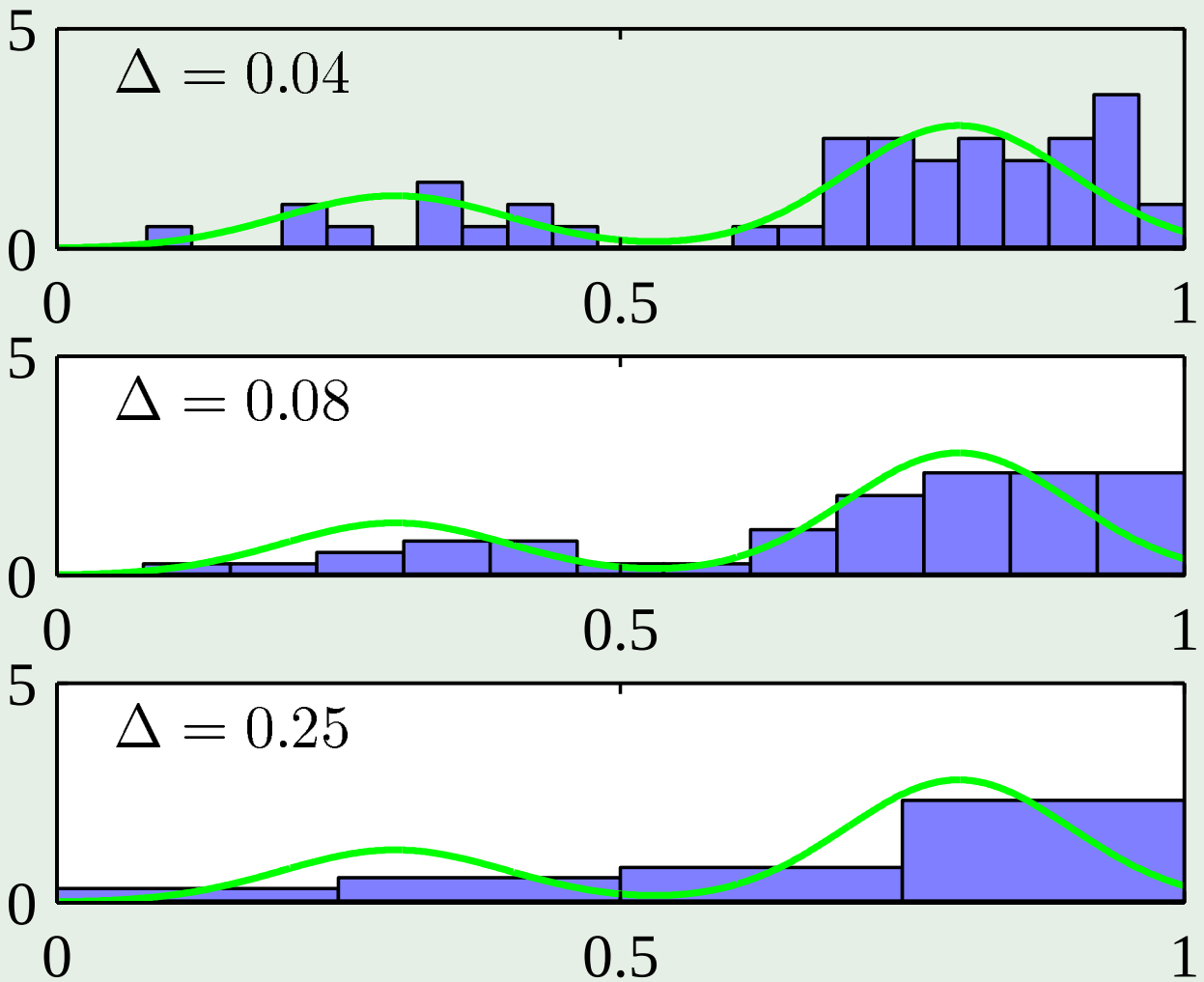
If we try to evaluate the distribution of data without assuming a parametric form of the density, we can use histograms:

- Discretise the data space
- Count the number of data elements in each bin

This has disadvantages:

- The bin positions and widths affect the density estimate
- The number of bins grows exponentially with the number of dimensions of the data

Example



Parzen Estimators

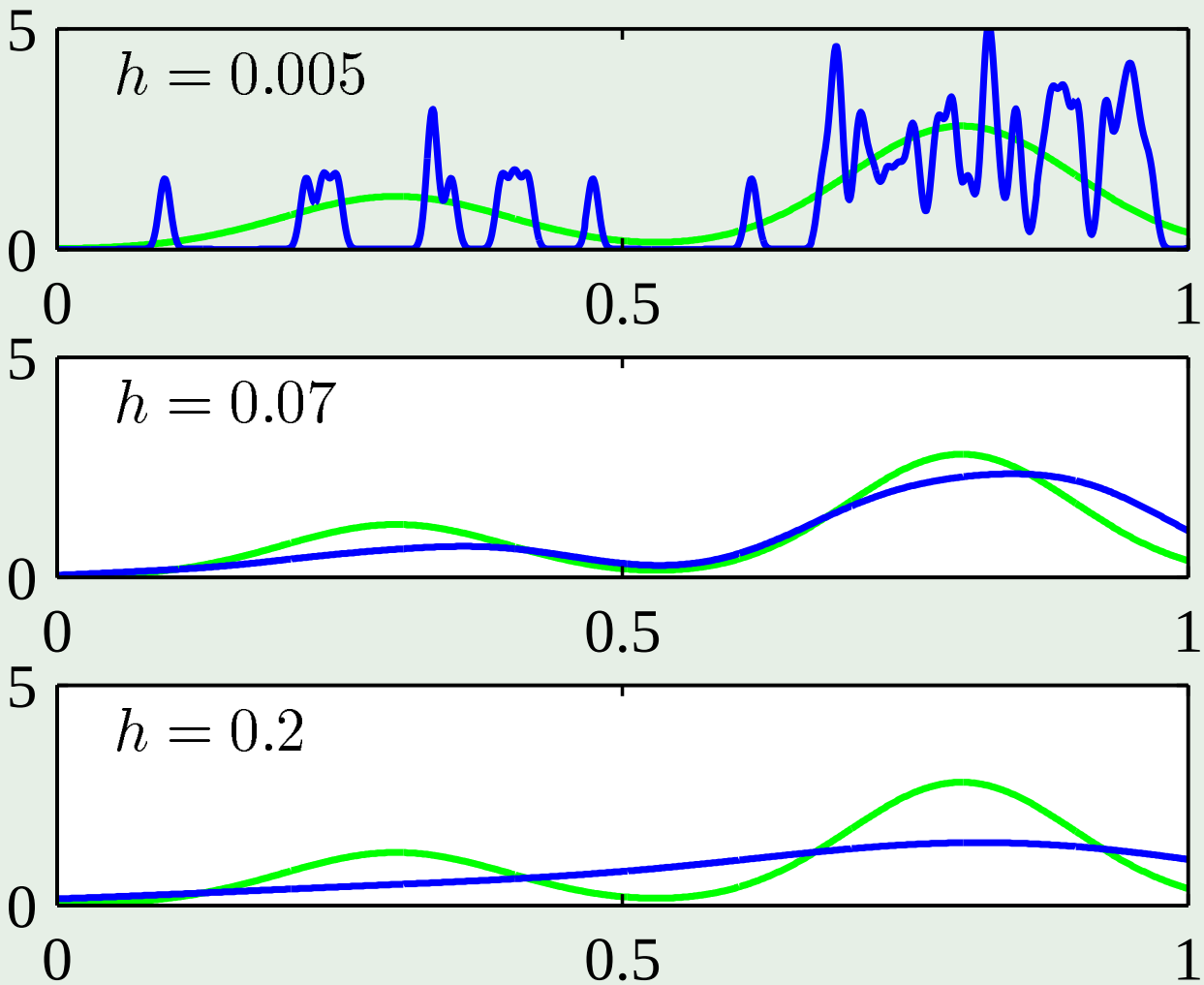
A technique that scales better with the dimensionality

- Consider a kernel function centred on each training data point
- If this kernel is non-negative for all $\mathbf{x}$ and normalised, this gives a valid estimate of the data density
- A common choice is the Gaussian. The corresponding estimate of the data density is then

$$p(\mathbf{x}) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{(2\pi h^2)^{1/2}} \exp\left(-\frac{||\mathbf{x} - \mathbf{x}_n||^2}{2h^2}\right) \qquad (1)$$

where h is the standard deviation of our Gaussian components and controls the smoothness of our estimate

Example



The Nadaraya-Watson Model

In this model, we use kernels to evaluate the joint distribution of $\mathbf{x}$ and t

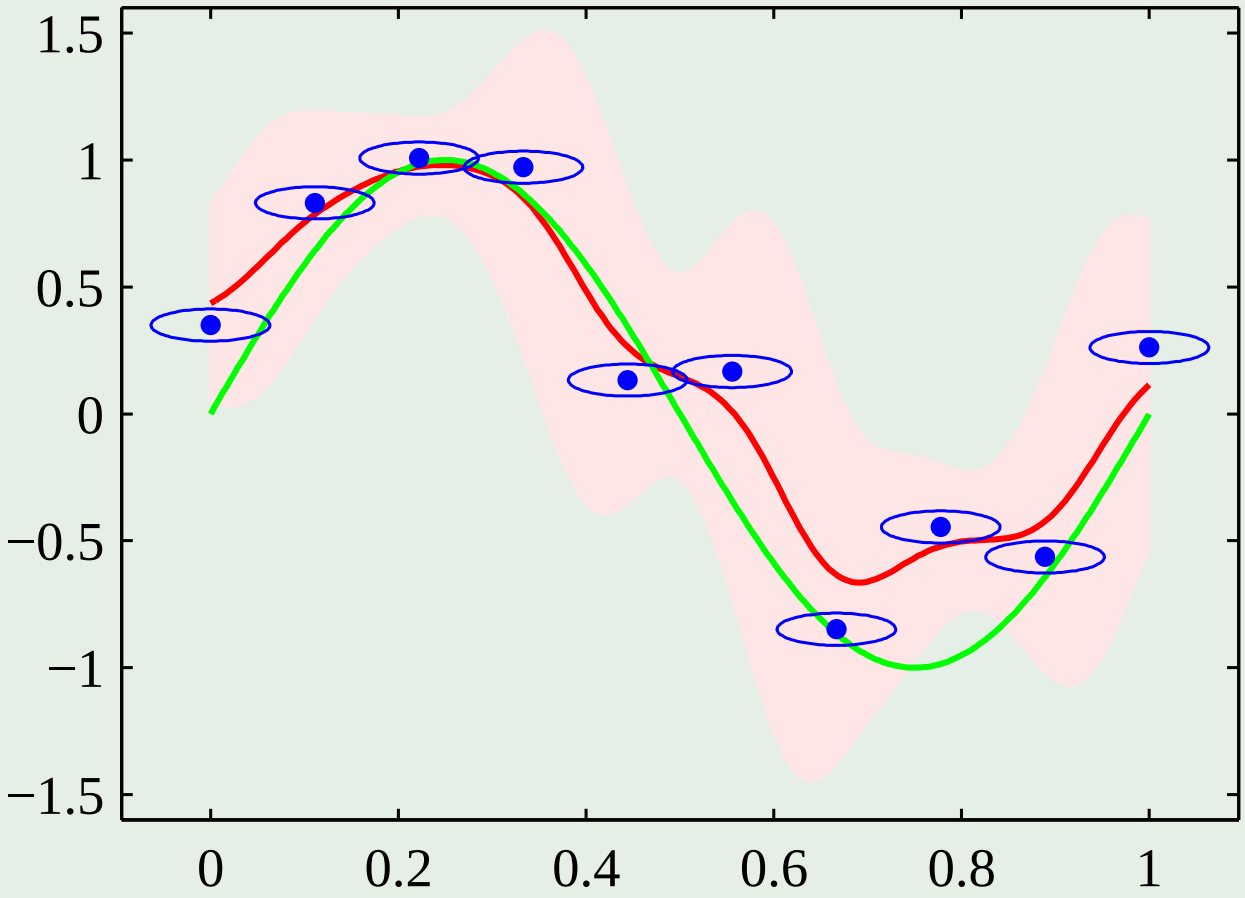
$$p(\mathbf{x}, t) = \frac{1}{N} \sum_{n=1}^{N} f(\mathbf{x} - \mathbf{x}_n, t - t_n) \tag{2}$$

We can then evaluate the conditional probability density

$$p(t|\mathbf{x}) = \frac{p(\mathbf{x}, t)}{\int p(\mathbf{x}, t)} \tag{3}$$

$$= \frac{\sum_{n=1}^{N} f(\mathbf{x} - \mathbf{x}_n, t - t_n)}{\sum_{m=1}^{N} \int f(\mathbf{x} - \mathbf{x}_m, t - t_m) dt} \tag{4}$$

Example



Kernel functions

Many linear, parametric models can be re-cast into the evaluation of a *kernel function* $k(\mathbf{x}, \mathbf{x}')$ at the training data points.

- This is called the dual representation

For models based on a fixed mapping $\mathbf{x} \rightarrow \phi(\mathbf{x})$, this kernel can be written as:

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}') \tag{5}$$

This allows us to extend many methods:

- If the input vector $\mathbf{x}$ only appears in scalar products, we can replace that product with some other kernel function.
- This is called *Kernel Substitution* or the *Kernel Trick*

Working in Kernel Space

We can rewrite our kernelised machine to work with the kernel function only

- That is, $\phi(\mathbf{x})$ need never be considered explicitly
- This allows us to consider very large (even infinite) feature vectors $\phi(\mathbf{x})$, while keeping the computations tractable

A trivial example: 3D computation in 2D

Consider the feature vector

$$\phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \tag{6}$$

$$\phi(\mathbf{x})^\top \phi(\mathbf{z}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)(z_1^2, \sqrt{2}z_1z_2, z_2^2)^\top \tag{7}$$

$$= x_1^2z_1^2 + 2x_1z_1x_2z_2 + x_2^2z_2^2 \tag{8}$$

$$= (\mathbf{x}^\top \mathbf{z})^2 = k(\mathbf{x}, \mathbf{z}) \tag{9}$$

Building Kernel Functions

Valid kernel functions can be obtained in a variety of ways:

- Explicitly create the feature vector $\phi(\mathbf{x})$ and find the corresponding $k(\mathbf{x}, \mathbf{x}')$. This can be very difficult to do.
- A sufficient condition for a kernel function is that the Gram Matrix $\mathbf{K}$

$$\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^\top = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix} \tag{10}$$

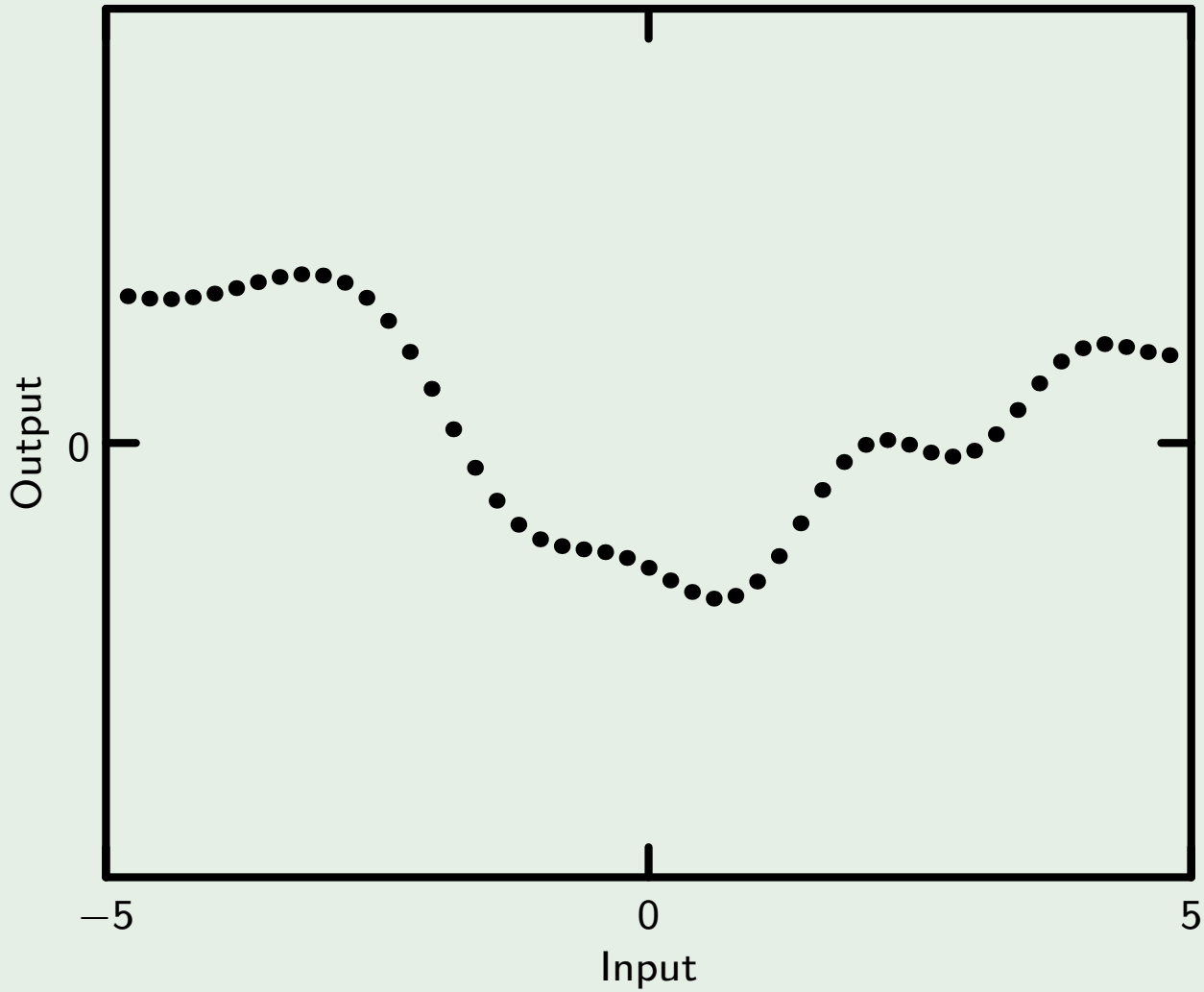
  is positive semi-definite for any set $\{\mathbf{x}_1 \ldots \mathbf{x}_n\}$
- The easiest way of constructing new kernel functions is to take a valid kernel and apply a transformation that is known to preserve positive semi-definiteness of $\mathbf{K}$

Gaussian Processes

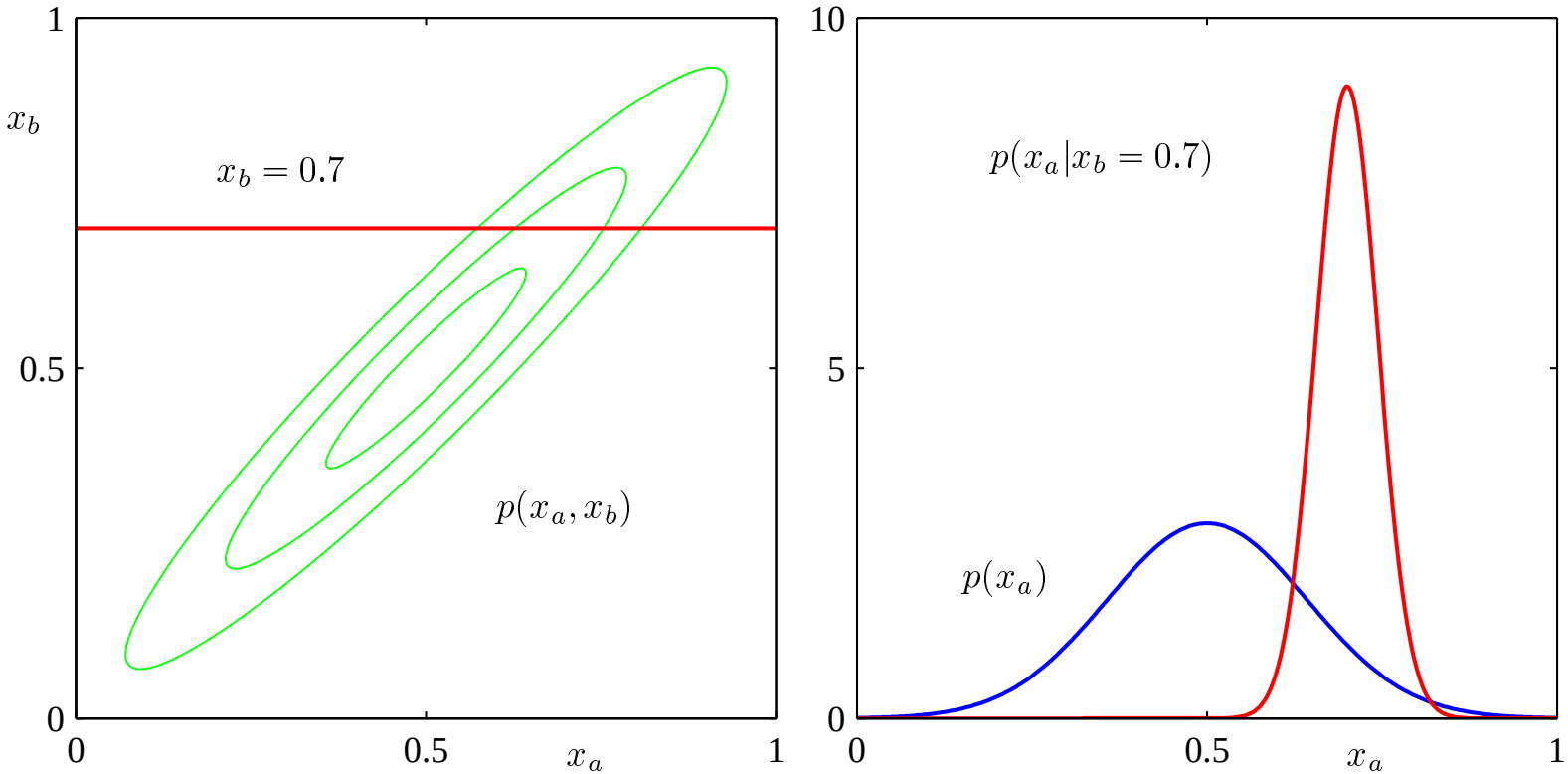
Core idea

- A Gaussian process defines a distribution over functions
- A function can be seen as an infinite set of values, one for each possible input
- In practice, we *never* need to evaluate the function for all possible inputs
- Therefore, it is sufficient to be *able* to evaluate the probability distribution over the outputs for any possible input.
- For every set of inputs, we need to be able to specify how the outputs covary.

Example: a sample from a Gaussian Process



The Gaussian Distribution



We consider the linear regression model

$$y(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) \tag{11}$$

with a zero-mean Gaussian prior over the weights

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|0, \alpha^{-1}\mathbf{I}) \tag{12}$$

This induces a probability distribution over the vector $\mathbf{y}$ of values of $y(\mathbf{x}_n)$, which is also Gaussian because $\mathbf{y} = \mathbf{\Phi w}$ The mean and covariance of $\mathbf{y}$ are

$$\mathbb{E}[\mathbf{y}] = \mathbf{\Phi}\mathbb{E}[\mathbf{w}] = 0 \tag{13}$$

$$\text{cov}[\mathbf{y}] = \mathbb{E}[\mathbf{y}\mathbf{y}^\top] = \mathbf{\Phi}\mathbb{E}[\mathbf{w}\mathbf{w}^\top]\mathbf{\Phi}^\top = \frac{1}{\alpha}\mathbf{\Phi}\mathbf{\Phi}^\top = \mathbf{K} \tag{14}$$

Gaussian Processes

In general, a Gaussian Process is defined as

- A distribution over $y(\mathbf{x})$,
- such that the joint distribution of a set of $y(\mathbf{x})$ at any set $\{\mathbf{x}_1 \ldots \mathbf{x}_N\}$ is Gaussian
- The process is defined solely by the mean and covariance
- The mean is generally taken to be zero
  - This is equivalent with choosing zero mean on the prior $p(\mathbf{w})$

Gaussian Processes for Regression

We define a model of our output values $\mathbf{y}$, which are normally distributed with as covariance some Gramm matrix $\mathbf{K}$

$$p(\mathbf{y}|\mathbf{X}) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K}) \tag{15}$$

NB: $\mathbf{K}$ is a function of the (training) data $\mathbf{x}_1 \dots \mathbf{x}_N$
If we consider that the target values $\mathbf{t}$ are normally i.i.d. around the output values, their conditional distribution is given by

$$p(\mathbf{t}|\mathbf{y}) = \mathcal{N}(\mathbf{t}|\mathbf{y}, \beta^{-1}\mathbf{I}) \tag{16}$$

The marginal distribution of the targets is then given by

$$p(\mathbf{t}|\mathbf{X}) = \int p(\mathbf{t}|\mathbf{y})p(\mathbf{y}|\mathbf{X})d\mathbf{y} = \mathcal{N}(\mathbf{t}|\mathbf{0}, \mathbf{C}) \tag{17}$$

where

$$C(\mathbf{x}_n, \mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) + \beta^{-1}\delta_{nm} \tag{18}$$

Prediction

The Gaussian Process then allows us to predict t_{N+1} given the training data $\mathbf{X}_N = (\mathbf{x}_1, \ldots, \mathbf{x}_N), \mathbf{t}_N = (t_1, \ldots, t_N)$ and a new input $\mathbf{x}_{N+1}$. We know the joint distribution over $\mathbf{t}_{N+1} = (t_1, \ldots, t_N, t_{N+1})$ is

$$p(\mathbf{t}_{N+1}|\mathbf{X}_{N+1}) = \mathcal{N}(\mathbf{t}_{N+1}|\mathbf{0}, \mathbf{C}_{N+1}) \tag{19}$$

It is then a standard result for the Gaussian that we can compute $p(t_{N+1}|\mathbf{t}_N) = \mathcal{N}(t_{N+1}|m, \sigma)$ by partitioning

$$\mathbf{C}_{N+1} = \begin{pmatrix} \mathbf{C}_N & \mathbf{k} \\ \mathbf{k}^\top & c \end{pmatrix} \tag{20}$$

and computing $m = \mathbf{k}^\top \mathbf{C}_N^{-1} \mathbf{t}_N$ and $\sigma = c - \mathbf{k}^\top \mathbf{C}_N^{-1} \mathbf{k}$

To summarise

1. We choose a prior over the possible functions that generated the data, in the form of a kernel function
2. We *marginalise out* the actual function and compute conditional the distribution of new targets based on the training data

Example

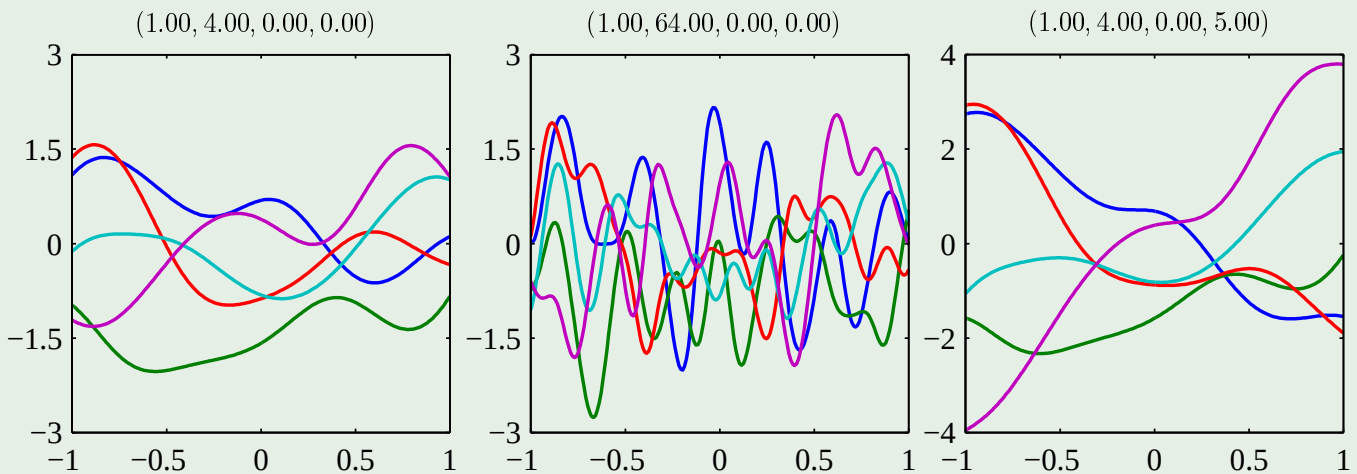
A usual prior is

$$k(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp\left(-\frac{\theta_1}{2}||\mathbf{x}_n - \mathbf{x}_m||^2\right) + \theta_2 + \theta_3 \mathbf{x}_n^\top \mathbf{x}_m \tag{21}$$

To summarise

1. We choose a prior over the possible functions that generated the data, in the form of a kernel function
2. We *marginalise out* the actual function and compute conditional the distribution of new targets based on the training data

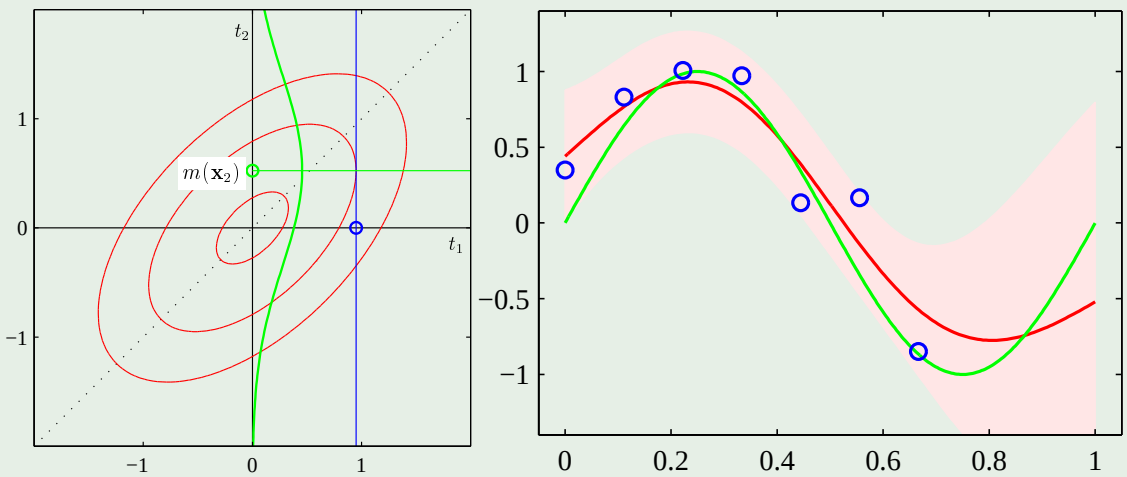
Example: Sampling from the prior



To summarise

1. We choose a prior over the possible functions that generated the data, in the form of a kernel function
2. We *marginalise out* the actual function and compute conditional the distribution of new targets based on the training data

Example: Conditioning on the data



Some other things to consider

- The prior has an important impact on the functions the model considers: finding good parameters for the kernel function is important
- The kernel function parameters can be learnt by maximum likelihood: Type 2 Maximum Likelihood
- Neural networks tend to Gaussian processes in the limit of $M \to \infty$ for a broad class of priors $p(\mathbf{w})$. However in this limit the hidden nodes are independent.
- Gaussian processes can be used for classification by transforming the output of the Gaussian process with a logistic function $\sigma(a)$. This requires approximations.

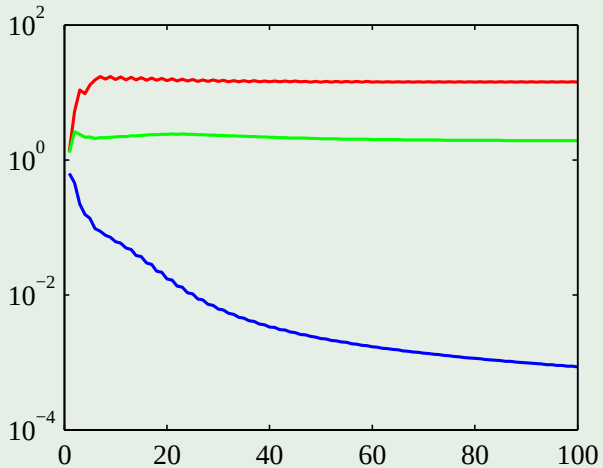
Automatic Relevance Determination (ARD)

Kernel functions can give different weights to different inputs.
Optimise the marginal likelihood to learn the value of each input.

$$k(\mathbf{x}, \mathbf{x}') = \theta_0 \exp\left(-\frac{1}{2}\exp\sum_{i=1}^{D}\eta_i(x_i - x_i')^2\right)$$

Example: Noisy variables

$$t \sim \mathcal{N}(sin(2\pi x_1)) \quad x_1 \sim \mathcal{N} \quad x_2 \sim \mathcal{N}(x_1) \quad x_3 \sim \mathcal{N}$$



Support vector machines

Kernel machines can be computationally expensive

- The kernel function $k(\mathbf{x}, \mathbf{x}')$ must be evaluated for all pairs $\mathbf{x}$ and $\mathbf{x}'$ of training points
- This can be computationally infeasible during training and lead to slow prediction for new data points
- One solution is to use only a small subset of the training data
- So how do we choose the training points to use?

Linear Classification

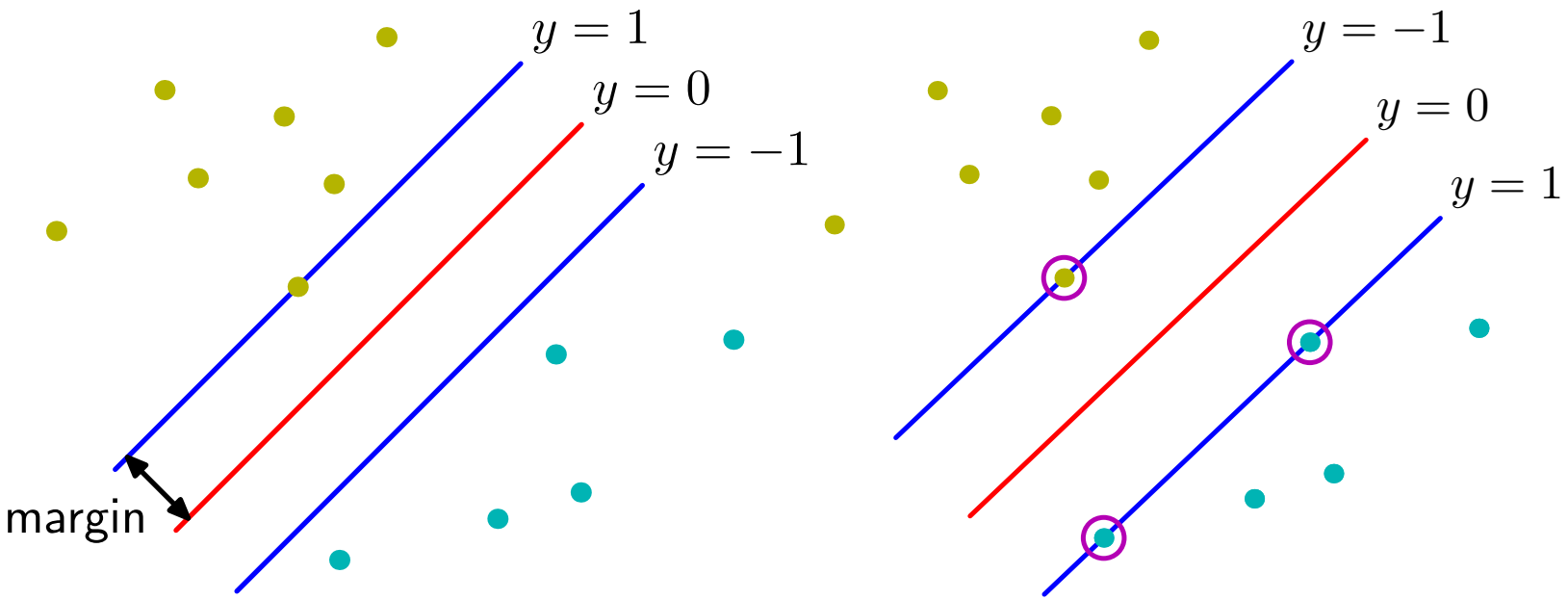
Consider a data set $(\mathbf{X}, \mathbf{t})$ where the targets $t_1, \ldots, t_N \in \{-1, +1\}$ are linearly separable.

- There exists at least one choice of parameters $\mathbf{w}$ so that

$$y(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b \text{ and } \begin{cases} y(\mathbf{x}_n) > 0 & \text{for all } t_n = +1 \\ y(\mathbf{x}_n) < 0 & \text{for all } t_n = -1 \end{cases}$$

- In order to optimise generalisation, we maximise the *margin*
- One justification for this is to describe the density of the data using Parzen estimators and find the hyperplane that minimises the probability of error. In the limit for Parzen estimators with $\sigma \to 0$ this hyperplane corresponds to the maximum margin separator.

Maximising the margin



Optimisation

This is a constrained optimisation problem: maximise the distance between the closest points and the margin, while keeping everything correctly classified:

$$\arg\max_{\mathbf{w},b} \left(\frac{1}{||\mathbf{w}||} \min_n [t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b)] \right) \tag{21}$$

This is fiendishly difficult, but we can make it easier: it is equivalent to minimising $||\mathbf{w}||^2$ subject to the constraints

$$t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b) \geqslant 1 \tag{22}$$

which we can solve using Lagrange multipliers.

The Lagrangian is given by:

$$\mathcal{L}(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{n=1}^{N} a_n \left(t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b) - 1 \right) \tag{23}$$

Solving this for $\mathbf{w}$ and b gives

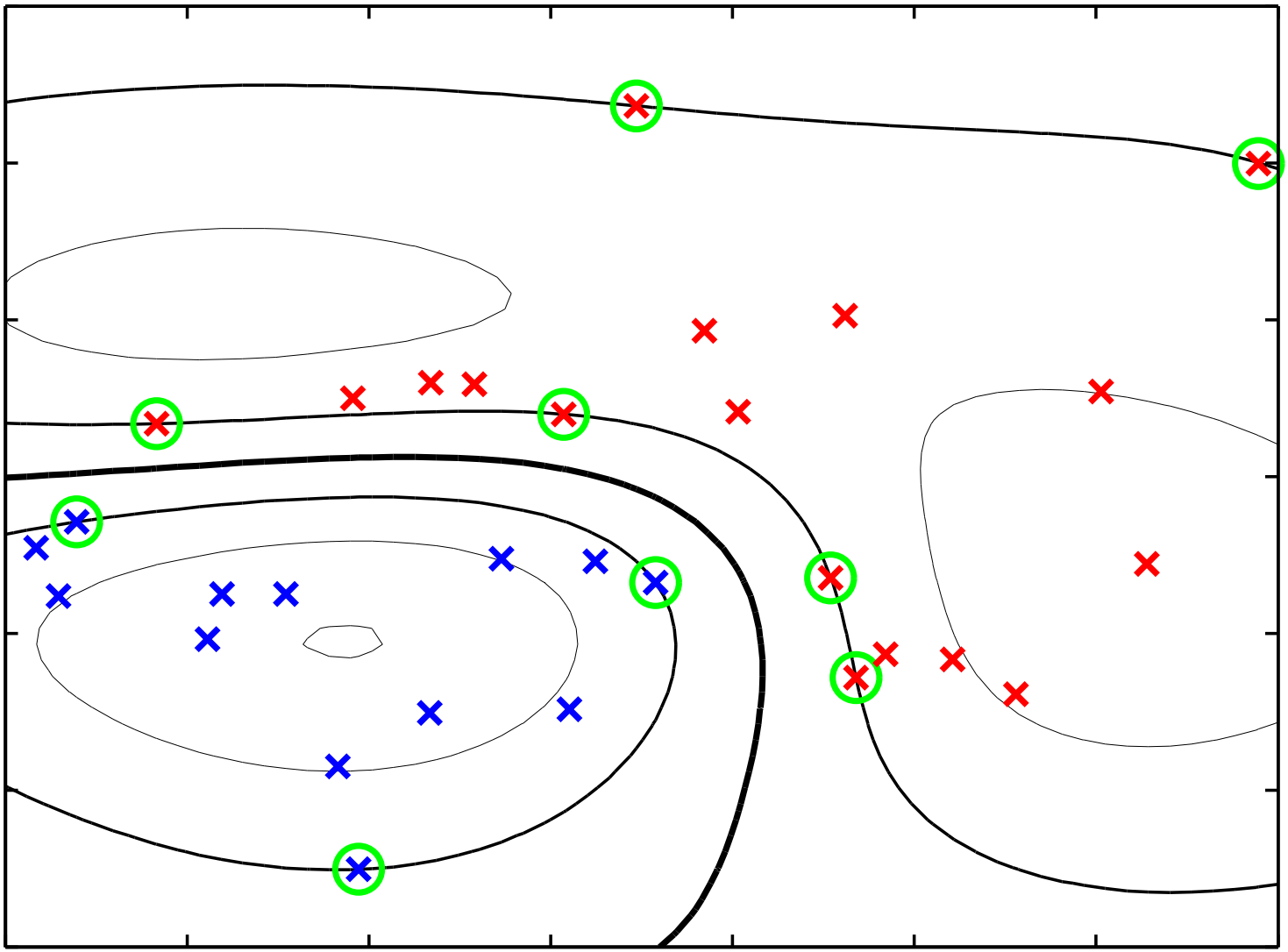
$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n) \qquad 0 = \sum_{n=1}^{N} a_n t_n \tag{24}$$

When we re-introduce these in the Lagrangian, we obtain the *dual representation* of the problem:

$$\tilde{\mathcal{L}}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \tag{25}$$

where $k(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_m)$ and from which we get $\mathbf{a}$

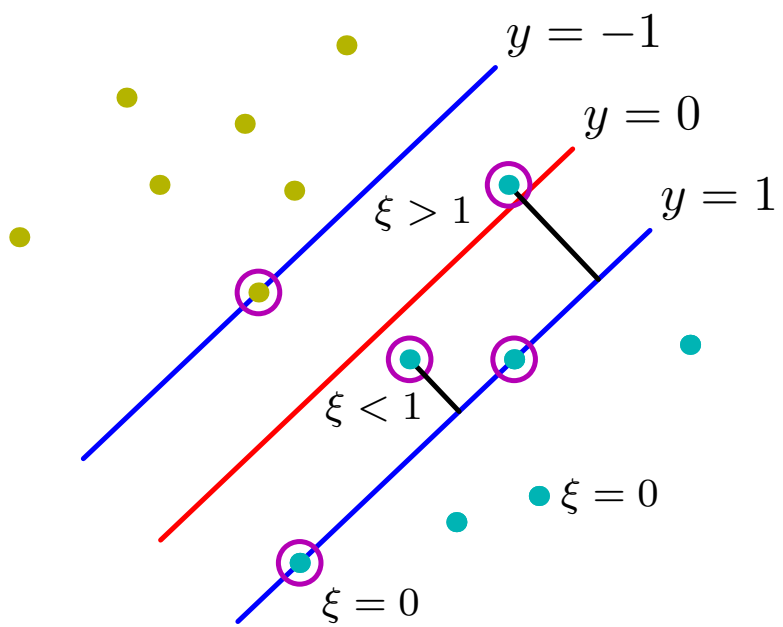
Classification with SVM



Other considerations

Some important aspects of SVM:

- We can extend the SVM to overlapping class distributions by introducing a slack variable ξ_n for each data point, and add a soft penalty for misclassified points



Other considerations

Some important aspects of SVM:

- We can extend the SVM to overlapping class distributions by introducing a slack variable ξ_n for each data point, and add a soft penalty for misclassified points
- SVM are inherently 2-class classifiers. We have seen before what problems occur when combining such classifiers for multi-class problems. This is still an open issue, but in practice combined one-versus-the-rest is most commonly used.
- SVMs only provide classifications, and no confidence in the classification. It is therefore difficult to integrate them with probabilistic methods.
- They are probably the best off-the-shelf classifier available at this point in time

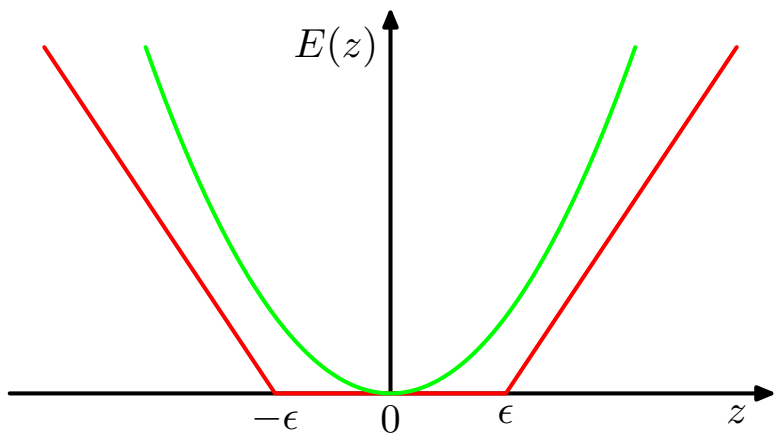
Regression

SVM can also be used for regression

$$E = \frac{1}{2}\sum_{n=1}^{N}(\mathbf{y}_n - t_n)^2 + \frac{\lambda}{2}||\mathbf{w}||^2 \longrightarrow C\sum_{n=1}^{N} E_\epsilon(\mathbf{y}_n - t_n)^2 + \frac{1}{2}||\mathbf{w}||^2$$

with

$$E_\epsilon(y-t) = \begin{cases} 0 & \text{if } |y-t| < \epsilon \\ |y-t| - \epsilon & \text{otherwise.} \end{cases}$$



Summary

We've talked about kernel methods

- Dual representation (Bishop, p. 291-294)
- Parzen estimators (Bishop, p. 122-124)
- Gaussian Processes (Bishop, p. 303-313)
- Support Vector Machines (Bishop, p.325-331)

For the lab, we continue our implementation of EM and next week work with GP and SVM.