Lecture 6

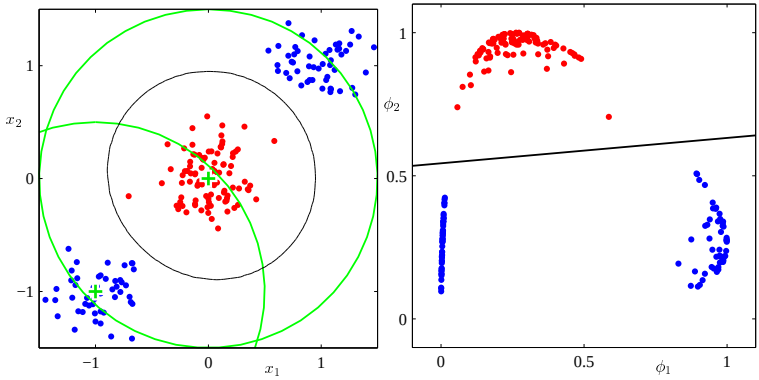
Support Vector Machines

G. Englebienne
M. Poel

University of Twente

Basis Functions

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



Parametric vs. Non-parametric

We've seen 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

We've seen 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

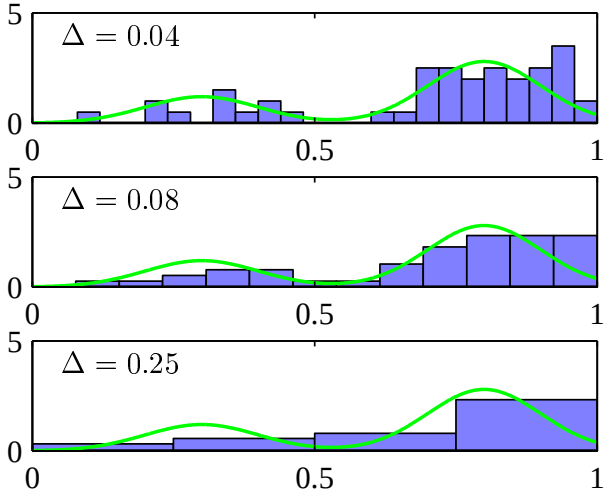
If we try to evaluate the distribution of data without assuming a parametric form of the density, we can use histograms:

- 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

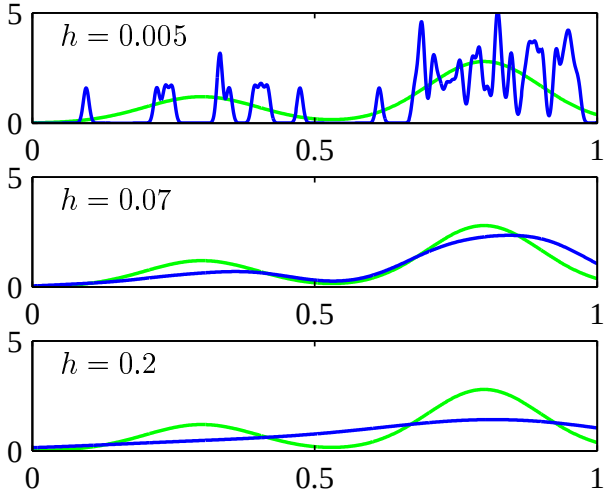
A technique that scales better with the dimensionality

- 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

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

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

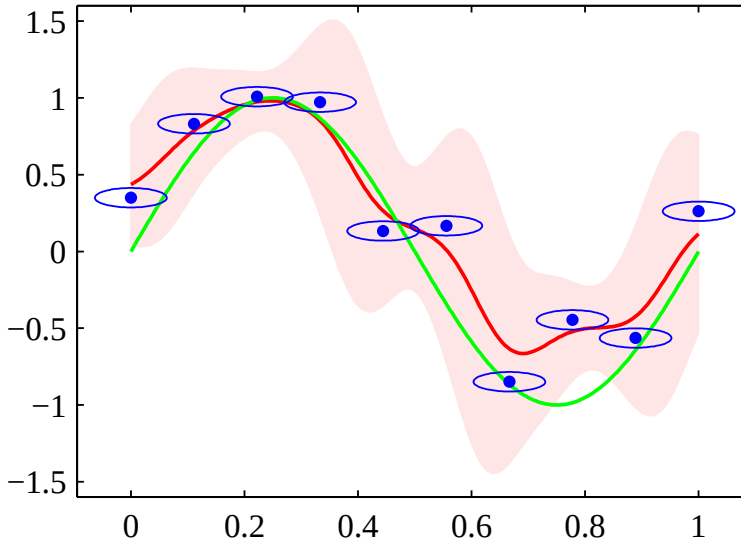
$$p(\mathbf{x}, t) = \frac{1}{N} \sum_{n=1}^{N} f(\mathbf{x} - \mathbf{x}_n, t - t_n) \tag{2}$$

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

$$\begin{aligned}
\phi(\mathbf{x}) &= (x_1^2, \sqrt{2}x_1x_2, x_2^2) \\
\phi(\mathbf{x})^\top \phi(\mathbf{z}) &= (x_1^2, \sqrt{2}x_1x_2, x_2^2)(z_1^2, \sqrt{2}z_1z_2, z_2^2)^\top \\
&= x_1^2z_1^2 + 2x_1z_1x_2z_2 + x_2^2z_2^2 \\
&= (x_1z_1 + x_2z_2)^2 \\
&= (\mathbf{x}^\top \mathbf{z})^2 = k(\mathbf{x}, \mathbf{z})
\end{aligned}$$

Some Kernel Functions

- Polynomials of degree q: $K(\mathbf{x}^\top \mathbf{y} + 1)^q$ For degree 2:

$$\begin{aligned} K(\mathbf{x}, \mathbf{y}) &= (\mathbf{x}^\top \mathbf{y} + 1 + 1)^2 \\ &= (x_1 y_1 + x_2 y_2 + 1)^2 \\ &= 1 + 2x_1 y_1 + 2x_2 y_2 + 2x_1 x_2 y_1 y_2 + x_1^2 y_1^2 + x_2^2 y_2^2 \\ \phi(\mathbf{x}) &= [1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2, x_1^2, x_2^2]^\top \end{aligned}$$

- Radial-basis functions

$$K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{||\mathbf{x} - \mathbf{y}||^2}{\sigma^2}\right)$$

- Sigmoidal functions

$$K(\mathbf{x}, \mathbf{y}) = \tanh(2\mathbf{x}^\top \mathbf{y} + 1)$$

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 $\mathbf{K}$

$$\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^\top = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix} \tag{6}$$

  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 $\mathbf{K}$

- Constrained Optimisation: common problem
  - Example: Optimise the probablity under a model while respecting the axioms of probability
- General form: optimise a multivariate function

$$f(\mathbf{x}) \tag{7}$$

subject to a constraint that links the variables in vector $\mathbf{x}$. We can write such a constraint in the form:

$$g(\mathbf{x}) = 0 \tag{8}$$

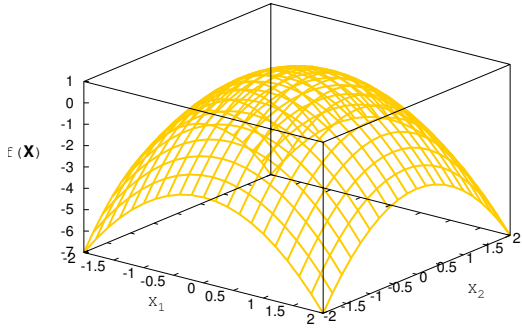
Example: Two-dimensional problem

Find $\mathbf{x}$ for which

$$f(\mathbf{x}) = 1 - x_1^2 - x_2^2 \tag{9}$$

is maximal, subject to the constraint

$$g(\mathbf{x}) = x_1 + x_2 - 1 = 0 \tag{10}$$



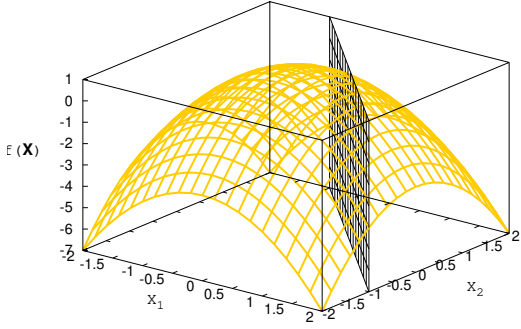
Example: Two-dimensional problem

Find $\mathbf{x}$ for which

$$f(\mathbf{x}) = 1 - x_1^2 - x_2^2 \tag{9}$$

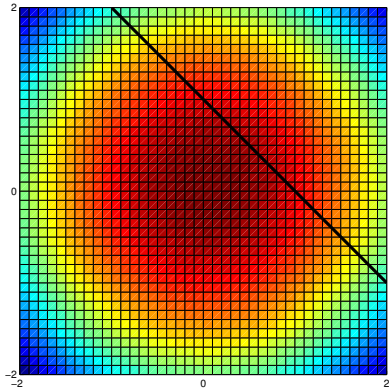
is maximal, subject to the constraint

$$g(\mathbf{x}) = x_1 + x_2 - 1 = 0 \tag{10}$$



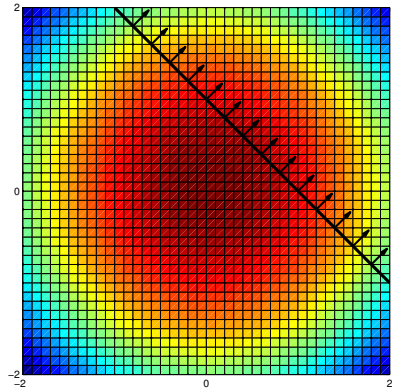
Example

- The constraint defines a $(D-1)$-dimensional manifold in the D-dimensional input space
- The Gradient of the constraint $\nabla g(\mathbf{x})$ is a normal to that manifold, in any point of the manifold



Example

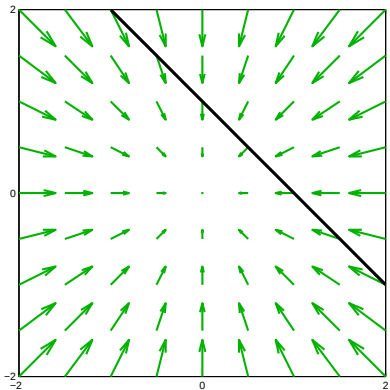
- The constraint defines a $(D-1)$-dimensional manifold in the D-dimensional input space
- The Gradient of the constraint $\nabla g(\mathbf{x})$ is a normal to that manifold, in any point of the manifold



Gradients

- The gradient of the function at the optimum must be perpendicular to the manifold of the constraint:
  - If the gradient was not normal to the manifold, we could improve the function value by following the gradient
- So the optimum is where $\nabla f(\mathbf{x})$ and $\nabla g(\mathbf{x})$ are parallel to each other

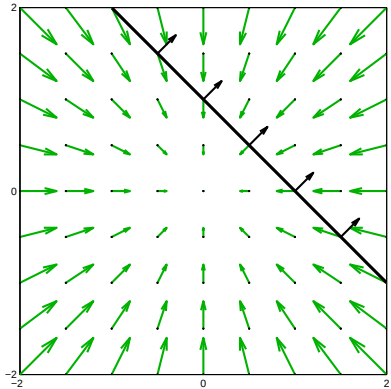
Example



Gradients

- The gradient of the function at the optimum must be perpendicular to the manifold of the constraint:
  - If the gradient was not normal to the manifold, we could improve the function value by following the gradient
- So the optimum is where $\nabla f(\mathbf{x})$ and $\nabla g(\mathbf{x})$ are parallel to each other

Example



Lagrangian

- The gradients are parallel, but can have different lengths (and different signs)
- The constrained maximum of the function is therefore where

$$\nabla f(\mathbf{x}) + \lambda \nabla g(\mathbf{x}) = 0 \tag{11}$$

- It is convenient to introduce the *L*agrangian function:

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x}) \tag{12}$$

- The optimum is found where the gradient $\nabla_{\mathbf{x}} L = 0$ and $\frac{\partial}{\partial \lambda} L = 0$

Example

Our function was:

$$f(\mathbf{x}) = 1 - x_1^2 - x_2^2 \tag{13}$$

and the constraint was written as

$$g(\mathbf{x}) = x_1 + x_2 - 1 = 0. \tag{14}$$

The corresponding Lagrangian function is thus

$$L(\mathbf{x}, \lambda) = 1 - x_1^2 - x_2^2 + \lambda(x_1 + x_2 - 1) \tag{15}$$

Example

Taking the gradient and partial derivative of the Lagrangian function

$$L(\mathbf{x}, \lambda) = 1 - x_1^2 - x_2^2 + \lambda(x_1 + x_2 - 1) \tag{16}$$

results in the following set of equations

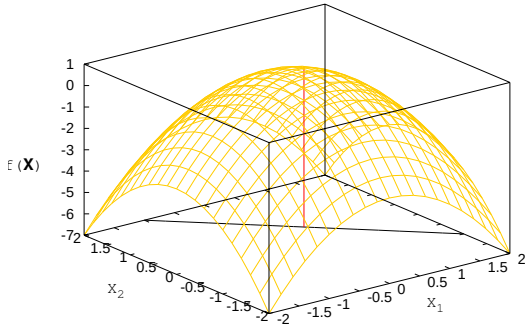
$$\begin{cases} -2x_1 + \lambda = 0 \\ -2x_2 + \lambda = 0 \\ x_1 + x_2 - 1 = 0 \end{cases} \tag{17}$$

Which we solve to obtain

$$\mathbf{x}^* = (\tfrac{1}{2}, \tfrac{1}{2}) \tag{18}$$

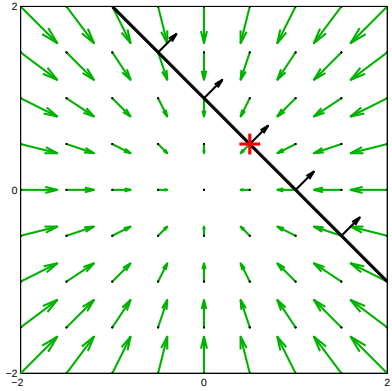
Worked-out example

Example



Worked-out example

Example

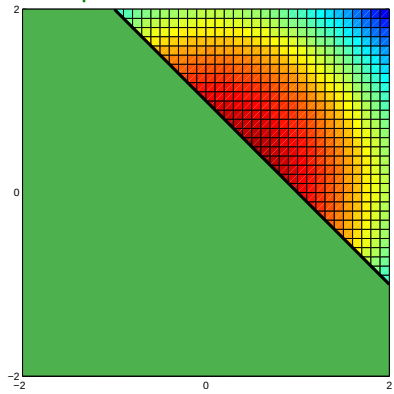


Inequality constraints

Instead of using constraints of the form $g(\mathbf{x}) = 0$, we now consider inequality constraints of the form $g(\mathbf{x}) \geqslant 0$.

The constraint can now be active or inactive

Example

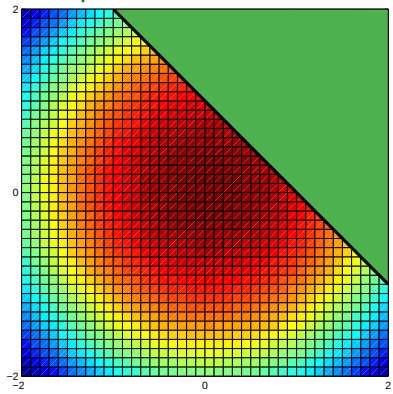


Inequality constraints

Instead of using constraints of the form $g(\mathbf{x}) = 0$, we now consider inequality constraints of the form $g(\mathbf{x}) \geqslant 0$.

The constraint can now be active or inactive

Example



Active versus inactive

In the inactive case, the optimum is simply given by

$$\nabla f(\mathbf{x}) = 0 \tag{19}$$

This is a solution of the Lagrangian function when $\lambda = 0$
If the constraint is active, the solution lies on the boundary as before.

- This is the case when $\lambda \neq 0$.
- The sign of the gradient is now crucial!

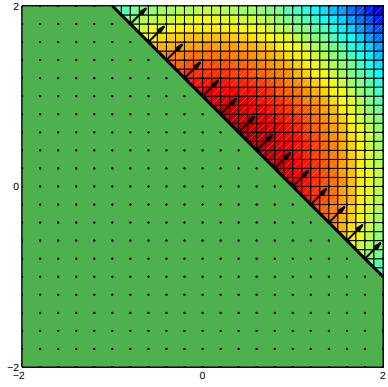
Example

- Active:

$$g(\mathbf{x}) = x_1 + x_2 - 1 \geqslant 0$$

- Inactive:

$$g(\mathbf{x}) = -x_1 - x_2 + 1 \geqslant 0$$



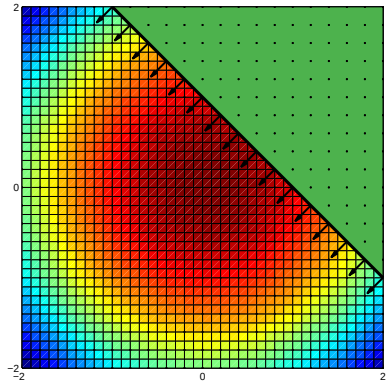
Example

- Active:

$$g(\mathbf{x}) = x_1 + x_2 - 1 \geqslant 0$$

- Inactive:

$$g(\mathbf{x}) = -x_1 - x_2 + 1 \geqslant 0$$



Solving inequality constraints

The function $f(\mathbf{x})$ is optimised subject to the constraint $g(\mathbf{x}) \geqslant 0$ when either $g(\mathbf{x}) = 0$ (active case) or $\lambda = 0$ (inactive case), where the gradients have opposite signs:

$$\nabla f(\mathbf{x}) = -\lambda \nabla g(\mathbf{x}) \quad \lambda \geqslant 0 \tag{20}$$

The solution is therefore obtained when

$$\begin{cases} g(\mathbf{x}) & \geqslant 0 \\ \lambda & \geqslant 0 \\ \lambda g(\mathbf{x}) & = 0 \end{cases} \tag{21}$$

These are the *Karush-Kuhn-Tucker* (KKT) equations.

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?

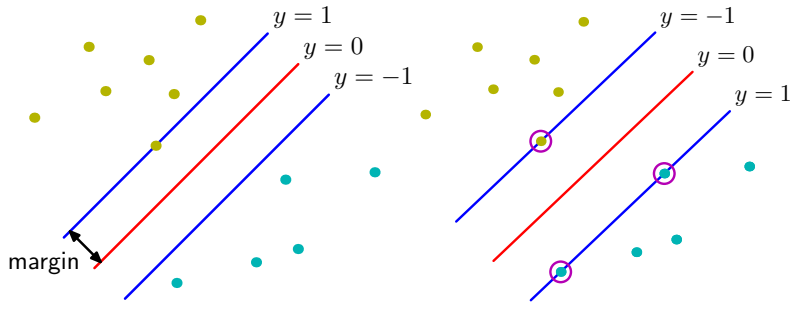
Consider a data set $(\mathbf{X}, \mathbf{t})$ where the targets $t_1, \ldots, t_N \in \{-1, +1\}$ are linearly separable.

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

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

- 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



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

$$\mathbf{w}^\top \phi(\mathbf{x}) + b \geqslant 0 \text{ for all } t_n = +1$$
$$\mathbf{w}^\top \phi(\mathbf{x}) + b \leqslant 0 \text{ for all } t_n = -1$$

which can be rewritten as

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

Maximise the margin

- The margin: distance from discriminant to closest instances
- Distance from $\mathbf{x}$ to hyperplane is

$$\frac{|\mathbf{w}^\top \mathbf{x}_n + b|}{||\mathbf{w}||}$$

- To maximise the distance, we minimise $||\mathbf{w}||^2$ subject to

$$\frac{t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b)}{||\mathbf{w}||} \geqslant \rho \quad \forall n$$

- To avoid underconstrainedness, we fix $\rho||\mathbf{w}|| = 1$, leading to the *primal formulation*:

$$\min \frac{1}{2}||\mathbf{w}||^2 \text{ subject to } t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b) \geqslant 1, \forall n$$

Solving the SVM boundary

The Lagrangian is given by:

$$\mathcal{L}(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{n=1}^{N} a_n \left(t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b) - 1 \right) \tag{22}$$

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

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n) \qquad 0 = \sum_{n=1}^{N} a_n t_n \tag{23}$$

So $\mathbf{w}$ is a linear combination of (training) data points

Solving the SVM boundary

The Lagrangian is given by:

$$\mathcal{L}(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{n=1}^{N} a_n \left(t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b) - 1 \right) \tag{22}$$

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

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n) \qquad 0 = \sum_{n=1}^{N} a_n t_n \tag{23}$$

So $\mathbf{w}$ is a linear combination of (training) data points

Dual representation

When we re-introduce the solution for $\mathbf{w}$ in the Lagrangian, we obtain the *dual representation* of the problem:

$$\begin{aligned}
\mathcal{L}(\mathbf{w}, b, \mathbf{a}) &= \frac{1}{2}\mathbf{w}^\top\mathbf{w} - \sum_{n=1}^{N} a_n \left(t_n(\mathbf{w}^\top\phi(\mathbf{x}_n) + b) - 1 \right) \\
&= \frac{1}{2}\mathbf{w}^\top\mathbf{w} - \mathbf{w}^\top \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n) - b\sum_{n=1}^{N} a_n t_n + \sum_{n=1}^{N} a_n \\
&= -\frac{1}{2}\mathbf{w}^\top\mathbf{w} + \sum_{n=1}^{N} a_n \\
&= \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m \phi(\mathbf{x}_n)^\top\phi(\mathbf{x}_m)
\end{aligned}$$

subject to $\sum_n a_n t_n = 0$ and $a_n \geqslant 0 \quad \forall n$

Why the dual representation

The dual representation may seem counter-effectual:

- We had to learn $\mathbf{w}$, d parameters, in the primal representation
- We need to learn N parameters a_n in the dual
- We now need to store the training data

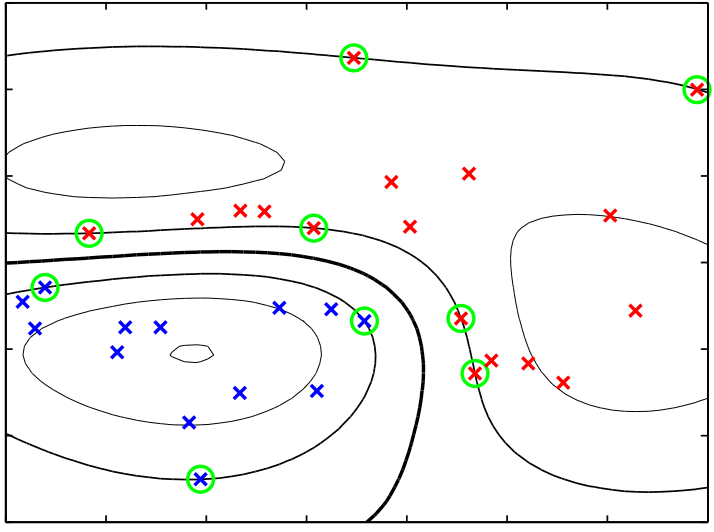
However, as we hinted in the beginning:

- Most $a_n = 0$ (only the support vectors have non-zero a_n and need to be stored
- If $d >> N$, this is more efficient
- The dual form only involves $\mathbf{x}_i^\top \mathbf{x}_k$, thus allowing kernelisation

Kernelisation

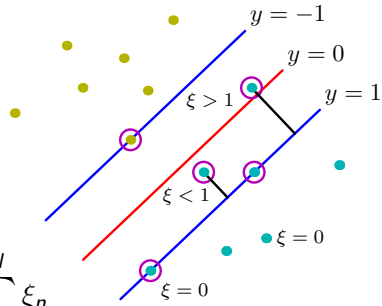
- In the dual representation, $\phi(\mathbf{x}_n)$ appears only as an inner product of data points
  - Kernel trick: $\phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_m) \rightarrow K(\mathbf{x}_n, \mathbf{x}_m)$
- Finding the solution of the dual can be done by quadratic programming
- Most a_n are zero ($\Rightarrow$ sparse solution)
- Those datapoints for which $a_n > 0$ are the support vectors

Classification with SVM



Soft margin

- Not linearly separable: $t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b) \geqslant 1 - \xi_n$
- Minimise the slack: $\sum_n \xi_n$



- The new primal is:

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

Other considerations

Some important aspects of SVM:

- SVM are inherently 2-class classifiers. In practice combining one-versus-the-rest classifiers is most commonly used.
- SVMs only provide classifications, and no confidence in the classification. It is therefore difficult to integrate them with probabilistic methods.
- One of the best off-the-shelf classifiers available at this point in time
- Can also be used for regression

Summary

We've talked about kernel methods

- Kernel trick and Dual representation (Bishop, p. 291-294)
- Parzen estimators (Bishop, p. 122-124)
- Lagrange multipliers (Bishop, p. 707-710)
- Support Vector Machines (Bishop, p. 325-331)

For the lab, we continue working on neural networks.

However...

Mannes and I are both unavailable coming Monday, so:

1. Next Monday's lecture (probabilistic modelling) will be given on Thursday (11/10)
2. Thursday's lab can be worked on on Monday
3. Locations might still change, check Canvas for announcements!