Kernel methods

Foundations of Artificial Intelligence

John Shawe-Taylor

j.shawe-taylor@ucl.ac.uk
Department of Computer Science
University College London

Overview

Worked example: Ridge Regression

Consider the problem of finding a homogeneous real-valued linear function

$$g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle = \mathbf{x}'\mathbf{w} = \sum_{i=1}^{n} w_i x_i,$$

that best interpolates a given set of 'training data':

$$S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$$

of points $\mathbf{x}_i$ from $X \subseteq \mathbb{R}^n$ with corresponding labels y_i in $Y \subseteq \mathbb{R}$. Note: $\mathbf{x}'$ denotes the transpose of $\mathbf{x}$, i.e. a row vector instead of a column vector.

Possible loss function

- Loss ℓ_g measures discrepancy between function output and correct output – squared to ensure always positive:

$$\ell_g((\mathbf{x}, y)) = (y - g(\mathbf{x}))^2$$

Possible loss function

- Loss ℓ_g measures discrepancy between function output and correct output – squared to ensure always positive:

$$\ell_g((\mathbf{x}, y)) = (y - g(\mathbf{x}))^2$$

- We introduce notation: matrix $\mathbf{X}$ has rows the m examples of S. Hence we can write

$$\xi = \mathbf{y} - \mathbf{X}\mathbf{w}$$

for the vector of differences between $g(\mathbf{x}_i)$ and y_i.

Optimising the choice of g

Need to ensure flexibility of g is controlled – controlling the norm of $\mathbf{w}$ proves effective:

$$\min_{\mathbf{w}} \mathcal{L}_\lambda(\mathbf{w}, S) = \min_{\mathbf{w}} \lambda \|\mathbf{w}\|^2 + \|\xi\|^2,$$

where we can compute

$$\begin{aligned} \|\xi\|^2 &= \langle \mathbf{y} - \mathbf{X}\mathbf{w}, \mathbf{y} - \mathbf{X}\mathbf{w} \rangle \\ &= \mathbf{y}'\mathbf{y} - 2\mathbf{w}'\mathbf{X}'\mathbf{y} + \mathbf{w}'\mathbf{X}'\mathbf{X}\mathbf{w} \end{aligned}$$

Setting derivative of $\mathcal{L}_\lambda(\mathbf{w}, S)$ equal to 0 gives

$$\mathbf{X}'\mathbf{X}\mathbf{w} + \lambda\mathbf{w} = \left(\mathbf{X}'\mathbf{X} + \lambda\mathbf{I}_n\right)\mathbf{w} = \mathbf{X}'\mathbf{y}$$

Primal solution

We get the primal solution weight vector:

$$\mathbf{w} = \left(\mathbf{X}'\mathbf{X} + \lambda \mathbf{I}_n\right)^{-1} \mathbf{X}'\mathbf{y}$$

and regression function

$$g(\mathbf{x}) = \mathbf{x}'\mathbf{w} = \mathbf{x}' \left(\mathbf{X}'\mathbf{X} + \lambda \mathbf{I}_n\right)^{-1} \mathbf{X}'\mathbf{y}$$

Role of the regularisation parameter λ

- if $\lambda \to 0$ we obtain least squares regression

Role of the regularisation parameter λ

- if $\lambda \to 0$ we obtain least squares regression
- but note that $\mathbf{X}'\mathbf{X}$ may not be invertible: typically use pseudo-inverse, a sort of partial regularisation

Role of the regularisation parameter λ

- if $\lambda \to 0$ we obtain least squares regression
- but note that $\mathbf{X}'\mathbf{X}$ may not be invertible: typically use pseudo-inverse, a sort of partial regularisation
- if dimension of feature space is small compared to the amount of training data, likely to be invertible, but for large dimensional feature spaces may be unstable

Role of the regularisation parameter λ

- if $\lambda \to 0$ we obtain least squares regression
- but note that $\mathbf{X}'\mathbf{X}$ may not be invertible: typically use pseudo-inverse, a sort of partial regularisation
- if dimension of feature space is small compared to the amount of training data, likely to be invertible, but for large dimensional feature spaces may be unstable
- as λ increases the solution becomes more stable, but at the expense of being 'damped'

Implications for design of ML systems

- if feature space too simple, we may not be able to represent the function we want to learn:

Implications for design of ML systems

- if feature space too simple, we may not be able to represent the function we want to learn:
  - do poorly on the training data

Implications for design of ML systems

- if feature space too simple, we may not be able to represent the function we want to learn:
  - do poorly on the training data
  - so-called *underfitting*

Implications for design of ML systems

- if feature space too simple, we may not be able to represent the function we want to learn:
  - do poorly on the training data
  - so-called *underfitting*
- but if we create more complex features we may create unstable learning

Implications for design of ML systems

- if feature space too simple, we may not be able to represent the function we want to learn:
  - do poorly on the training data
  - so-called *underfitting*
- but if we create more complex features we may create unstable learning
  - solution becomes dependent on the particular (chance) choice of training data

Implications for design of ML systems

- if feature space too simple, we may not be able to represent the function we want to learn:
  - do poorly on the training data
  - so-called *underfitting*
- but if we create more complex features we may create unstable learning
  - solution becomes dependent on the particular (chance) choice of training data
  - regularisation with λ can potentially mitigate this danger

Implications for design of ML systems

- if feature space too simple, we may not be able to represent the function we want to learn:
  - do poorly on the training data
  - so-called *underfitting*
- but if we create more complex features we may create unstable learning
  - solution becomes dependent on the particular (chance) choice of training data
  - regularisation with λ can potentially mitigate this danger
- is there a general way of making more complex feature spaces, while enabling good regularisation?

Recall primal solution

We get the primal solution weight vector:

$$\mathbf{w} = \left(\mathbf{X}'\mathbf{X} + \lambda \mathbf{I}_n\right)^{-1} \mathbf{X}'\mathbf{y}$$

and regression function

$$g(\mathbf{x}) = \mathbf{x}'\mathbf{w} = \mathbf{x}' \left(\mathbf{X}'\mathbf{X} + \lambda \mathbf{I}_n\right)^{-1} \mathbf{X}'\mathbf{y}$$

Dual solution

A dual solution should involve only computation of inner products – this is achieved by expressing the weight vector as a linear combination of the training examples:

$$\mathbf{X}'\mathbf{X}\mathbf{w} + \lambda\mathbf{w} = \mathbf{X}'\mathbf{y} \quad \text{implies}$$

$$\mathbf{w} = \frac{1}{\lambda}\left(\mathbf{X}'\mathbf{y} - \mathbf{X}'\mathbf{X}\mathbf{w}\right) = \mathbf{X}'\frac{1}{\lambda}\left(\mathbf{y} - \mathbf{X}\mathbf{w}\right) = \mathbf{X}'\alpha,$$

where

$$\alpha = \frac{1}{\lambda}\left(\mathbf{y} - \mathbf{X}\mathbf{w}\right) \tag{1}$$

or equivalently

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i \mathbf{x}_i$$

Dual solution

Substituting $\mathbf{w} = \mathbf{X}'\alpha$ into equation (1) we obtain:

$$\lambda\alpha = \mathbf{y} - \mathbf{X}\mathbf{X}'\alpha$$

implying

$$\left(\mathbf{X}\mathbf{X}' + \lambda\mathbf{I}_m\right)\alpha = \mathbf{y}$$

This gives the dual solution:

$$\alpha = \left(\mathbf{X}\mathbf{X}' + \lambda\mathbf{I}_m\right)^{-1}\mathbf{y}$$

and regression function

$$g(\mathbf{x}) = \mathbf{x}'\mathbf{w} = \mathbf{x}'\mathbf{X}'\alpha = \sum_{i=1}^{m} \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle$$

Key ingredients of dual solution

Step 1: Compute

$$\alpha = (\mathbf{K} + \lambda \mathbf{I}_m)^{-1} \mathbf{y}$$

where $\mathbf{K} = \mathbf{X}\mathbf{X}'$ that is $\mathbf{K}_{ij} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$

Key ingredients of dual solution

Step 1: Compute

$$\alpha = (\mathbf{K} + \lambda \mathbf{I}_m)^{-1} \mathbf{y}$$

where $\mathbf{K} = \mathbf{X}\mathbf{X}'$ that is $\mathbf{K}_{ij} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$

Step 2: Evaluate on new point $\mathbf{x}$ by

$$g(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle$$

Key ingredients of dual solution

Step 1: Compute

$$\alpha = (\mathbf{K} + \lambda \mathbf{I}_m)^{-1} \mathbf{y}$$

where $\mathbf{K} = \mathbf{X}\mathbf{X}'$ that is $\mathbf{K}_{ij} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$

Step 2: Evaluate on new point $\mathbf{x}$ by

$$g(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle$$

Important observation: Both steps only involve inner products between input examples

Applying the 'kernel trick'

Since the computation only involves inner products, we can substitute for all occurrences of $\langle \cdot, \cdot \rangle$ by a function κ that computes:

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$$

and we obtain an algorithm for ridge regression in the feature space F defined by the mapping

$$\phi : \mathbf{x} \longmapsto \phi(\mathbf{x}) \in F$$

Note:

Applying the 'kernel trick'

Since the computation only involves inner products, we can substitute for all occurrences of $\langle \cdot, \cdot \rangle$ by a function κ that computes:

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$$

and we obtain an algorithm for ridge regression in the feature space F defined by the mapping

$$\phi : \mathbf{x} \longmapsto \phi(\mathbf{x}) \in F$$

Note:

- if ϕ is the identity, κ is the standard inner product and this corresponds to the primal case

Applying the 'kernel trick'

Since the computation only involves inner products, we can substitute for all occurrences of $\langle \cdot, \cdot \rangle$ by a function κ that computes:

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$$

and we obtain an algorithm for ridge regression in the feature space F defined by the mapping

$$\phi : \mathbf{x} \longmapsto \phi(\mathbf{x}) \in F$$

Note:

- if ϕ is the identity, κ is the standard inner product and this corresponds to the primal case
- the function κ is known as a *kernel function* and is of particular interest when there are short-cuts in its computation.

A simple kernel example

The simplest non-trivial kernel function is the quadratic kernel:

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle^2$$

involving just one extra operation. But surprisingly this kernel function now corresponds to a complex feature mapping:

$$\begin{aligned} \kappa(\mathbf{x}, \mathbf{z}) &= (\mathbf{x}'\mathbf{z})^2 = \mathbf{z}'(\mathbf{x}\mathbf{x}')\mathbf{z} \\ &= \langle \mathrm{vec}(\mathbf{z}\mathbf{z}'), \mathrm{vec}(\mathbf{x}\mathbf{x}') \rangle \end{aligned}$$

where $\mathrm{vec}(A)$ stacks the columns of the matrix A on top of each other. Hence, κ computes the inner product in the space defined by the feature mapping

$$\phi : \mathbf{x} \longmapsto \mathrm{vec}(\mathbf{x}\mathbf{x}')$$

Implications of the kernel trick

- Consider for example computing a regression function over 1000 images represented by pixel vectors – say $32 \times 32 = 1024$.

Implications of the kernel trick

- Consider for example computing a regression function over 1000 images represented by pixel vectors – say $32 \times 32 = 1024$.
- By using the quadratic kernel we implement the regression function in an approximately 1,000,000 dimensional space

Implications of the kernel trick

- Consider for example computing a regression function over 1000 images represented by pixel vectors – say $32 \times 32 = 1024$.
- By using the quadratic kernel we implement the regression function in an approximately 1,000,000 dimensional space
- but actually using less computation for the learning phase than we did in the original space as we need to invert a 1000×1000 matrix, rather than a 1024×1024 matrix.

Implications of the kernel trick

- Consider for example computing a regression function over 1000 images represented by pixel vectors – say $32 \times 32 = 1024$.
- By using the quadratic kernel we implement the regression function in an approximately 1,000,000 dimensional space
- but actually using less computation for the learning phase than we did in the original space as we need to invert a 1000×1000 matrix, rather than a 1024×1024 matrix.
- The evaluation phase will nonetheless be more expensive as will involve computing the kernel between test point and each of the 1000 training examples, rather than one inner product between weight vector and test point.

Overview

Implications of kernel algorithms

- Can perform linear regression in very high-dimensional (even infinite dimensional) spaces efficiently.

Implications of kernel algorithms

- Can perform linear regression in very high-dimensional (even infinite dimensional) spaces efficiently.
- This is equivalent to performing non-linear regression in the original input space: for example quadratic kernel leads to solution of the form

$$g(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i \langle \mathbf{x}_i, \mathbf{x} \rangle^2$$

that is a quadratic polynomial function of the components of the input vector $\mathbf{x}$.

Implications of kernel algorithms

- Can perform linear regression in very high-dimensional (even infinite dimensional) spaces efficiently.
- This is equivalent to performing non-linear regression in the original input space: for example quadratic kernel leads to solution of the form

$$g(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i \langle \mathbf{x}_i, \mathbf{x} \rangle^2$$

  that is a quadratic polynomial function of the components of the input vector $\mathbf{x}$.
- Using these high-dimensional spaces must surely come with a health warning, what about the curse of dimensionality?

Kernel methods approach

- Data embedded into a Euclidean feature space

Kernel methods approach

- Data embedded into a Euclidean feature space
- Linear relations are sought among the images of the data

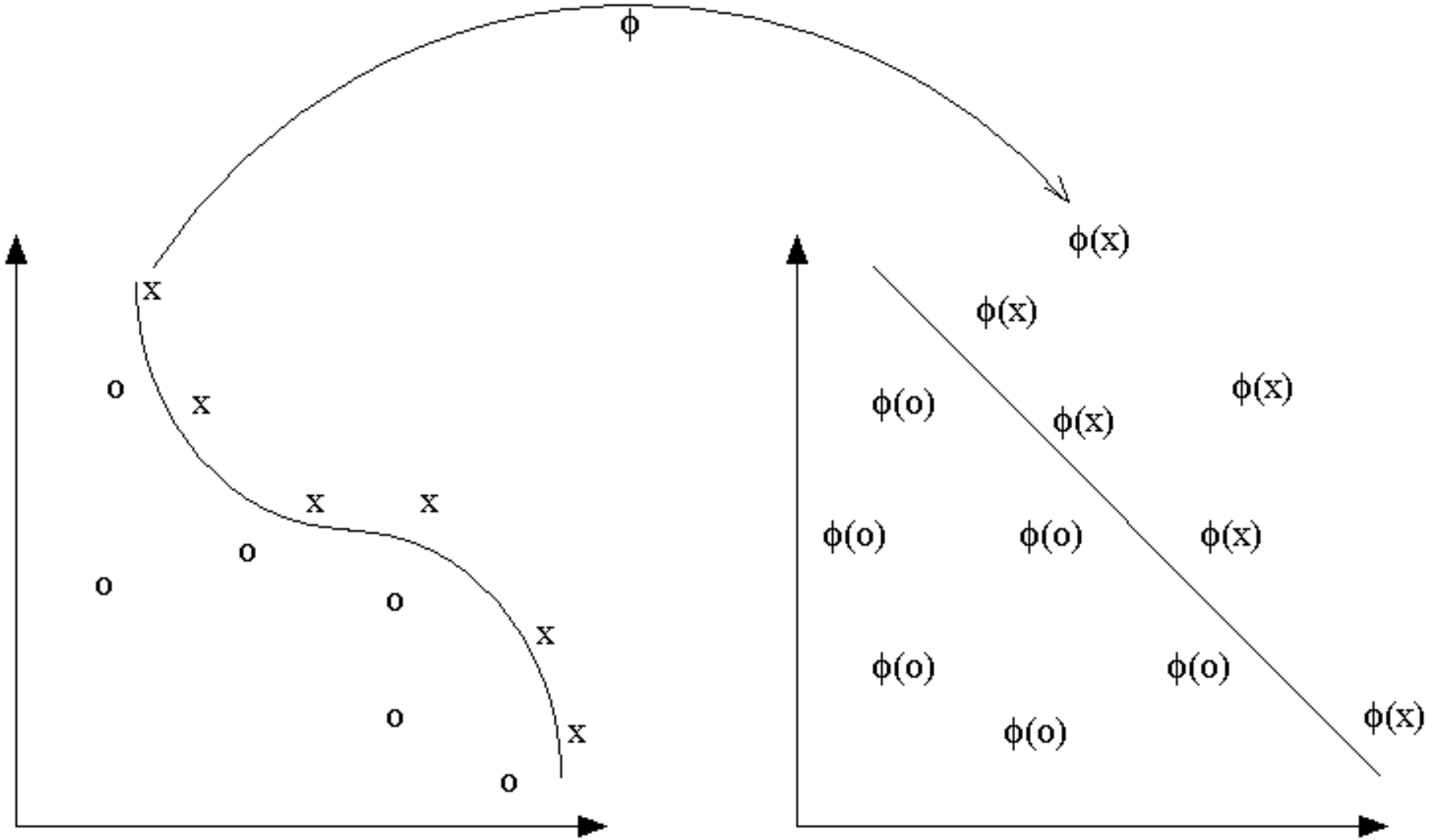
Kernel methods approach

- Data embedded into a Euclidean feature space
- Linear relations are sought among the images of the data
- Algorithms implemented so that only require inner products between vectors

Kernel methods approach

- Data embedded into a Euclidean feature space
- Linear relations are sought among the images of the data
- Algorithms implemented so that only require inner products between vectors
- Embedding designed so that inner products of images of two points can potentially be computed directly by an efficient ‘short-cut’ known as the kernel.

Kernel methods approach



Some properties of kernels

- kernels are symmetric:

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle = \langle \phi(\mathbf{z}), \phi(\mathbf{x}) \rangle = \kappa(\mathbf{z}, \mathbf{x})$$

Some properties of kernels

- kernels are symmetric:

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle = \langle \phi(\mathbf{z}), \phi(\mathbf{x}) \rangle = \kappa(\mathbf{z}, \mathbf{x})$$

- kernel matrices are positive semi-definite (psd):

$$\begin{aligned}
\mathbf{u}'\mathbf{K}\mathbf{u} &= \sum_{i,j=1}^{m} u_i u_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle \\
&= \left\langle \sum_{i=1}^{m} u_i \phi(\mathbf{x}_i), \sum_{j=1}^{m} u_j \phi(\mathbf{x}_j) \right\rangle \\
&= \left\| \sum_{i=1}^{m} u_i \phi(\mathbf{x}_i) \right\|^2 \geqslant 0
\end{aligned}$$

Kernel functions

- These two properties are all that is required for a kernel function to be valid: symmetric and every kernel matrix is psd.

Kernel functions

- These two properties are all that is required for a kernel function to be valid: symmetric and every kernel matrix is psd.
- Note that this is equivalent to all eigenvalues non-negative for the inner product matrix of any set of data
  - recall that eigenvalues of the kernel matrix measure the sum of the squares of the projections onto the eigenvectors.

Kernel functions

- These two properties are all that is required for a kernel function to be valid: symmetric and every kernel matrix is psd.
- Note that this is equivalent to all eigenvalues non-negative for the inner product matrix of any set of data
  - recall that eigenvalues of the kernel matrix measure the sum of the squares of the projections onto the eigenvectors.
- If we have uncountable domains should also have continuity, though there are exceptions to this as well.

Kernel constructions

For κ_1, κ_2 valid kernels, ϕ any feature map, $\mathbf{B}$ psd matrix, $a \geqslant 0$ and f any real valued function, the following are valid kernels:

- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z}) + \kappa_2(\mathbf{x}, \mathbf{z})$,

Kernel constructions

For κ_1, κ_2 valid kernels, ϕ any feature map, $\mathbf{B}$ psd matrix, $a \geqslant 0$ and f any real valued function, the following are valid kernels:

- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z}) + \kappa_2(\mathbf{x}, \mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = a\kappa_1(\mathbf{x}, \mathbf{z})$,

Kernel constructions

For κ_1, κ_2 valid kernels, ϕ any feature map, $\mathbf{B}$ psd matrix, $a \geqslant 0$ and f any real valued function, the following are valid kernels:

- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z}) + \kappa_2(\mathbf{x}, \mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = a\kappa_1(\mathbf{x}, \mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z})\kappa_2(\mathbf{x}, \mathbf{z})$,

Kernel constructions

For κ_1, κ_2 valid kernels, ϕ any feature map, $\mathbf{B}$ psd matrix, $a \geqslant 0$ and f any real valued function, the following are valid kernels:

- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z}) + \kappa_2(\mathbf{x}, \mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = a\kappa_1(\mathbf{x}, \mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z})\kappa_2(\mathbf{x}, \mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})\kappa_1(\mathbf{x}, \mathbf{z})f(\mathbf{z})$,

Kernel constructions

For κ_1, κ_2 valid kernels, ϕ any feature map, $\mathbf{B}$ psd matrix, $a \geqslant 0$ and f any real valued function, the following are valid kernels:

- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z}) + \kappa_2(\mathbf{x}, \mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = a\kappa_1(\mathbf{x}, \mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z})\kappa_2(\mathbf{x}, \mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})\kappa_1(\mathbf{x}, \mathbf{z})f(\mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\phi(\mathbf{x}), \phi(\mathbf{z}))$,

Kernel constructions

For κ_1, κ_2 valid kernels, ϕ any feature map, $\mathbf{B}$ psd matrix, $a \geqslant 0$ and f any real valued function, the following are valid kernels:

- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z}) + \kappa_2(\mathbf{x}, \mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = a\kappa_1(\mathbf{x}, \mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z})\kappa_2(\mathbf{x}, \mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})\kappa_1(\mathbf{x}, \mathbf{z})f(\mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\phi(\mathbf{x}), \phi(\mathbf{z}))$,
- $\kappa(\mathbf{x}, \mathbf{z}) = \mathbf{x}'\mathbf{B}\mathbf{z}$.

Kernel constructions

For κ_1, κ_2 valid kernels, ϕ any feature map, $\mathbf{B}$ psd matrix, $a \geqslant 0$ and f any real valued function, the following are valid kernels:

- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z}) + \kappa_2(\mathbf{x}, \mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = a\kappa_1(\mathbf{x}, \mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z})\kappa_2(\mathbf{x}, \mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})\kappa_1(\mathbf{x}, \mathbf{z})f(\mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\phi(\mathbf{x}), \phi(\mathbf{z}))$,
- $\kappa(\mathbf{x}, \mathbf{z}) = \mathbf{x}'\mathbf{B}\mathbf{z}$.
- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{x})^{-1/2}\kappa_1(\mathbf{x}, \mathbf{z})\kappa_1(\mathbf{z}, \mathbf{z})^{-1/2}$, the normalised kernel,

Kernel constructions

Following are also valid kernels:

Kernel constructions

Following are also valid kernels:

- $\kappa(\mathbf{x}, \mathbf{z}) = p(\kappa_1(\mathbf{x}, \mathbf{z}))$, for p any polynomial with positive coefficients.

Kernel constructions

Following are also valid kernels:

- $\kappa(\mathbf{x}, \mathbf{z}) = p(\kappa_1(\mathbf{x}, \mathbf{z}))$, for p any polynomial with positive coefficients.
- $\kappa(\mathbf{x}, \mathbf{z}) = \exp(\kappa_1(\mathbf{x}, \mathbf{z}))$, as $\exp$ has polynomial expansion with positive coefficients,

Kernel constructions

Following are also valid kernels:

- $\kappa(\mathbf{x}, \mathbf{z}) = p(\kappa_1(\mathbf{x}, \mathbf{z}))$, for p any polynomial with positive coefficients.
- $\kappa(\mathbf{x}, \mathbf{z}) = \exp(\kappa_1(\mathbf{x}, \mathbf{z}))$, as exp has polynomial expansion with positive coefficients,
- $\kappa(\mathbf{x}, \mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2 / (2\sigma^2))$.
  Proof: normalise second kernel with $\kappa_1(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle / \sigma^2$:

$$\frac{\exp(\langle \mathbf{x}, \mathbf{z} \rangle / \sigma^2)}{\sqrt{\exp(\|\mathbf{x}\|^2 / \sigma^2) \exp(\|\mathbf{z}\|^2 / \sigma^2)}} = \exp\left(\frac{\langle \mathbf{x}, \mathbf{z} \rangle}{\sigma^2} - \frac{\langle \mathbf{x}, \mathbf{x} \rangle}{2\sigma^2} - \frac{\langle \mathbf{z}, \mathbf{z} \rangle}{2\sigma^2}\right)$$

$$= \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|^2}{2\sigma^2}\right).$$

Kernel methods

Kernel methods (re)introduced in 1990s with Support Vector Machines

Kernel methods

Kernel methods (re)introduced in 1990s with Support Vector Machines

- Linear functions but in high dimensional spaces equivalent to non-linear functions in the input space

Kernel methods

Kernel methods (re)introduced in 1990s with Support Vector Machines

- Linear functions but in high dimensional spaces equivalent to non-linear functions in the input space
- Statistical analysis showing large margin can overcome curse of dimensionality

Kernel methods

Kernel methods (re)introduced in 1990s with Support Vector Machines

- Linear functions but in high dimensional spaces equivalent to non-linear functions in the input space
- Statistical analysis showing large margin can overcome curse of dimensionality
- Extensions rapidly introduced for many other tasks other than classification, eg example of kernel ridge regression presented above

Means and distances

Suppose we are given a kernel function:

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$$

and a training set S what can we estimate?

Means and distances

Suppose we are given a kernel function:

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$$

and a training set S what can we estimate?

- Consider some vector

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i \phi(\mathbf{x}_i)$$

we have

$$\|\mathbf{w}\|^2 = \left\langle \sum_{i=1}^{m} \alpha_i \phi(\mathbf{x}_i), \sum_{j=1}^{m} \alpha_j \phi(\mathbf{x}_j) \right\rangle = \sum_{i,j=1}^{m} \alpha_i \alpha_j \kappa(\mathbf{x}_i, \mathbf{x}_j)$$

Means and distances

- Hence, we can normalise data in the feature space:

$$\phi(\mathbf{x}) \mapsto \hat{\phi}(\mathbf{x}) = \frac{\phi(\mathbf{x})}{\|\phi(\mathbf{x})\|}$$

since we can compute the corresponding kernel $\hat{\kappa}$ by

$$\hat{\kappa}(\mathbf{x}, \mathbf{z}) = \left\langle \frac{\phi(\mathbf{x})}{\|\phi(\mathbf{x})\|}, \frac{\phi(\mathbf{z})}{\|\phi(\mathbf{z})\|} \right\rangle = \frac{\kappa(\mathbf{x}, \mathbf{z})}{\sqrt{\kappa(\mathbf{x}, \mathbf{x})\kappa(\mathbf{z}, \mathbf{z})}}$$

Means and distances

- Given two vectors:

$$\mathbf{w}_a = \sum_{i=1}^{m} \alpha_i \phi(\mathbf{x}_i) \text{ and } \mathbf{w}_b = \sum_{i=1}^{m} \beta_i \phi(\mathbf{x}_i)$$

we have

$$\mathbf{w}_a - \mathbf{w}_b = \sum_{i=1}^{m} (\alpha_i - \beta_i) \phi(\mathbf{x}_i)$$

so we can compute the distance between $\mathbf{w}_a$ and $\mathbf{w}_b$ as

$$d(\mathbf{w}_a, \mathbf{w}_b) = \|\mathbf{w}_a - \mathbf{w}_b\|$$

Means and distances

- For example the norm of the mean of a sample is given by

$$\|\phi_S\| = \left\| \frac{1}{m} \sum_{i=1}^{m} \phi(\mathbf{x}_i) \right\| = \frac{1}{m} \sqrt{\mathbf{j}'\mathbf{K}\mathbf{j}}$$

where $\mathbf{j}$ is the all ones vector.

Means and distances

- For example the norm of the mean of a sample is given by

$$\|\phi_S\| = \left\| \frac{1}{m} \sum_{i=1}^{m} \phi(\mathbf{x}_i) \right\| = \frac{1}{m} \sqrt{\mathbf{j}'\mathbf{K}\mathbf{j}}$$

  where $\mathbf{j}$ is the all ones vector.

- Hence, average squared distance to the mean of a sample is (check this calculation!):

$$\begin{aligned} \hat{\mathbb{E}}[\|\phi(\mathbf{x}) - \phi_S\|^2] &= \frac{1}{m} \sum_{i=1}^{m} \kappa(\mathbf{x}_i, \mathbf{x}_i) - \langle \phi_S, \phi_S \rangle \\ &= \frac{1}{m} \mathrm{tr}(\mathbf{K}) - \frac{1}{m^2} \mathbf{j}'\mathbf{K}\mathbf{j} \end{aligned}$$

Means and distances

- Consider centering the sample, i.e. moving the origin to the sample mean: this will result in

$$\|\phi s\|^2 = \frac{1}{m^2}\mathbf{j}'\mathbf{K}\mathbf{j} = 0$$

in the new coordinate system, while the lhs of previous equation is unchanged by centering. Hence, centering minimises the trace.

Means and distances

- Consider centering the sample, i.e. moving the origin to the sample mean: this will result in

$$\|\phi_S\|^2 = \frac{1}{m^2}\mathbf{j}'\mathbf{K}\mathbf{j} = 0$$

  in the new coordinate system, while the lhs of previous equation is unchanged by centering. Hence, centering minimises the trace.
- Centering is achieved by transformation:

$$\phi(\mathbf{x}) \mapsto \hat{\phi}(\mathbf{x}) = \phi(\mathbf{x}) - \phi_S$$

Means and distances

- What is effect on kernel and kernel matrix?

$$\begin{aligned}\hat{\kappa}(\mathbf{x}, \mathbf{z}) &= \langle \hat{\phi}(\mathbf{x}), \hat{\phi}(\mathbf{z}) \rangle \\ &= \kappa(\mathbf{x}, \mathbf{z}) - \frac{1}{m} \sum_{i=1}^{m} (\kappa(\mathbf{x}, \mathbf{x}_i) + \kappa(\mathbf{z}, \mathbf{x}_i)) + \frac{1}{m^2} \mathbf{j}' \mathbf{K} \mathbf{j}\end{aligned}$$

Means and distances

- What is effect on kernel and kernel matrix?

$$\begin{aligned}\hat{\kappa}(\mathbf{x},\mathbf{z}) &= \langle\hat{\phi}(\mathbf{x}),\hat{\phi}(\mathbf{z})\rangle \\ &= \kappa(\mathbf{x},\mathbf{z}) - \frac{1}{m}\sum_{i=1}^{m}(\kappa(\mathbf{x},\mathbf{x}_i)+\kappa(\mathbf{z},\mathbf{x}_i)) + \frac{1}{m^2}\mathbf{j}'\mathbf{K}\mathbf{j}\end{aligned}$$

- Hence we can implement the centering of a kernel matrix by

$$\hat{\mathbf{K}} = \mathbf{K} - \frac{1}{m}(\mathbf{j}\mathbf{j}'\mathbf{K} + \mathbf{K}\mathbf{j}\mathbf{j}') + \frac{1}{m^2}(\mathbf{j}'\mathbf{K}\mathbf{j})\mathbf{j}\mathbf{j}'$$

Overview

Simple classification algorithm

- Consider finding the centres of mass of positive and negative examples and classifying a test point by measuring which is closest

$$h(\mathbf{x}) = \operatorname{sgn}\left(\|\phi(\mathbf{x}) - \phi_{S_-}\|^2 - \|\phi(\mathbf{x}) - \phi_{S_+}\|^2\right)$$

Simple classification algorithm

- Consider finding the centres of mass of positive and negative examples and classifying a test point by measuring which is closest

$$h(\mathbf{x}) = \operatorname{sgn}\left(\|\phi(\mathbf{x}) - \phi_{S_-}\|^2 - \|\phi(\mathbf{x}) - \phi_{S_+}\|^2\right)$$

- we can express as a function of kernel evaluations

$$h(\mathbf{x}) = \operatorname{sgn}\left(\frac{1}{m_+}\sum_{i=1}^{m_+}\kappa(\mathbf{x}, \mathbf{x}_i) - \frac{1}{m_-}\sum_{i=m_++1}^{m}\kappa(\mathbf{x}, \mathbf{x}_i) - b\right),$$

where

$$b = \frac{1}{2m_+^2}\sum_{i,j=1}^{m_+}\kappa(\mathbf{x}_i, \mathbf{x}_j) - \frac{1}{2m_-^2}\sum_{i,j=m_++1}^{m}\kappa(\mathbf{x}_i, \mathbf{x}_j)$$

Simple classification algorithm

- equivalent to dividing the space with a hyperplane perpendicular to the line half way between the two centres with vector given by

$$\mathbf{w} = \frac{1}{m^+} \sum_{i=1}^{m^+} \phi(\mathbf{x}_i) - \frac{1}{m^-} \sum_{i=m^+ +1}^{m} \phi(\mathbf{x}_i)$$

Simple classification algorithm

- equivalent to dividing the space with a hyperplane perpendicular to the line half way between the two centres with vector given by

$$\mathbf{w} = \frac{1}{m^+} \sum_{i=1}^{m^+} \phi(\mathbf{x}_i) - \frac{1}{m^-} \sum_{i=m^+ +1}^{m} \phi(\mathbf{x}_i)$$

- Good exercise to check these calculations!

Simple novelty detection

- Consider putting a ball round the centre of mass ϕ_S of radius sufficient to contain all the data: i.e. data point $\mathbf{x}$ is novel if

$$\|\phi(\mathbf{x}) - \phi_S\| > \max_{1 \leqslant i \leqslant m} \|\phi(\mathbf{x}_i) - \phi_S\|$$

Simple novelty detection

- Consider putting a ball round the centre of mass ϕ_S of radius sufficient to contain all the data: i.e. data point $\mathbf{x}$ is novel if

$$\|\phi(\mathbf{x}) - \phi_S\| > \max_{1 \leqslant i \leqslant m} \|\phi(\mathbf{x}_i) - \phi_S\|$$

- Give a kernel expression for this quantity.

Perceptron algorithm

- For a classification task with thresholded linear functions

$$\mathbf{x} \longrightarrow \operatorname{sgn}\left(\langle \mathbf{w}, \mathbf{x} \rangle\right)$$

there is a simple algorithm to learn the weight vector $\mathbf{w}$ known as the Perceptron algorithm

Perceptron algorithm

- For a classification task with thresholded linear functions

$$\mathbf{x} \longrightarrow \operatorname{sgn}\left(\langle \mathbf{w}, \mathbf{x} \rangle\right)$$

  there is a simple algorithm to learn the weight vector $\mathbf{w}$ known as the Perceptron algorithm
- It initialises $\mathbf{w}_0 = 0$, $t = 0$ and then at each iteration it selects a training example j from

$$S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$$

  testing if the point $\mathbf{x}_j$ is correctly classified, i.e. if

$$y_j \left\langle \mathbf{w}_t, \mathbf{x}_j \right\rangle > 0$$

  if not correct, t is incremented, with $\mathbf{w}_t = \mathbf{w}_{t-1} + y_j \mathbf{x}_j$.

Kernel perceptron algorithm

- Since the weight vector is a linear combination of the training data it is straightforward to create a dual version of the algorithm, by writing

$$\mathbf{w}_t = \sum_{i=1}^{m} \alpha_i^t \mathbf{x}_i$$

Kernel perceptron algorithm

- Since the weight vector is a linear combination of the training data it is straightforward to create a dual version of the algorithm, by writing

$$\mathbf{w}_t = \sum_{i=1}^{m} \alpha_i^t \mathbf{x}_i$$

- The initialisation is then $\alpha^0 = 0$ with the test for correct classification

$$y_j \sum_{i=1}^{m} \alpha_i^t \left\langle \mathbf{x}_i, \mathbf{x}_j \right\rangle > 0$$

if not, t is incremented and $\alpha_j^t = \alpha_j^{t-1} + y_j$ with $\alpha_i^t = \alpha_i^{t-1}$ for all $i \neq j$.

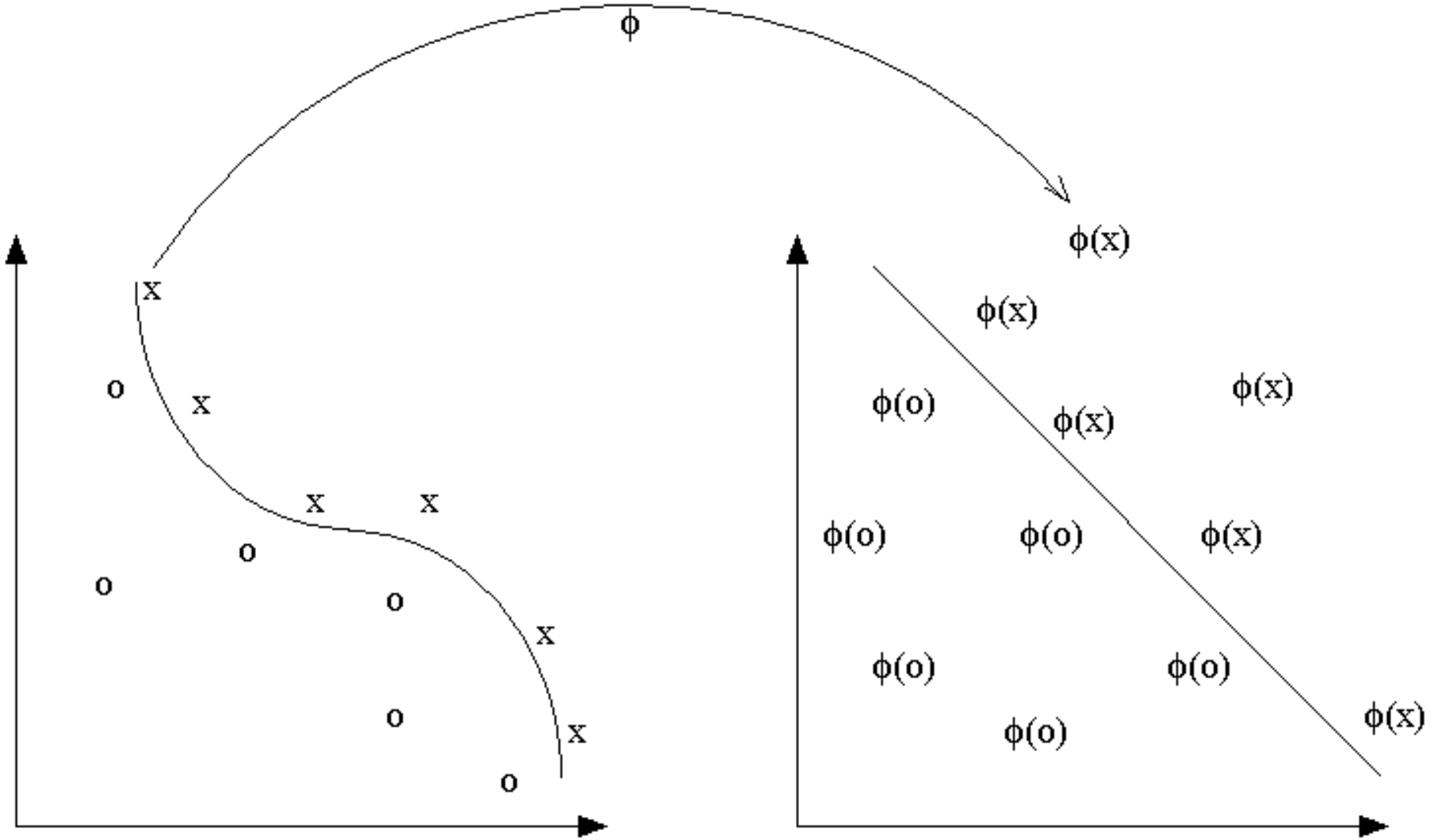
Kernel perceptron algorithm

- Hence, as with kernel Ridge Regression, we can run the algorithm with any kernel $\kappa(\mathbf{x}, \mathbf{z})$ in place of the inner product.

Kernel perceptron algorithm

- Hence, as with kernel Ridge Regression, we can run the algorithm with any kernel $\kappa(\mathbf{x}, \mathbf{z})$ in place of the inner product.
- This corresponds to running the perceptron algorithm in the feature space corresponding to the kernel κ.

Kernel methods approach



Novikoff theorem

- The performance of the Perceptron algorithm can be analysed using Novikoff's theorem.

Novikoff theorem

- The performance of the Perceptron algorithm can be analysed using Novikoff's theorem.
- It assumes there exists a weight vector $\mathbf{w}$ with $\|\mathbf{w}\| = 1$ such that

$$y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \geqslant \gamma > 0$$

for all i and that $\|\mathbf{x}_i\| \leqslant R$ for all i.

Novikoff theorem

- The performance of the Perceptron algorithm can be analysed using Novikoff's theorem.
- It assumes there exists a weight vector $\mathbf{w}$ with $\|\mathbf{w}\| = 1$ such that

$$y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \geqslant \gamma > 0$$

  for all i and that $\|\mathbf{x}_i\| \leqslant R$ for all i.
- With these assumptions Novikoff shows that there are at most

$$\frac{R^2}{\gamma^2}$$

  updates to $\mathbf{w}_t$

Novikoff for the Kernel Perceptron

- Novikoff applies to the kernel Perceptron algorithm with the assumption of the existence of a weight vector in the kernel defined feature space and the value of R given by

$$R = \max_i \sqrt{\kappa(\mathbf{x}_i, \mathbf{x}_i)}$$

Novikoff for the Kernel Perceptron

- Novikoff applies to the kernel Perceptron algorithm with the assumption of the existence of a weight vector in the kernel defined feature space and the value of R given by

$$R = \max_i \sqrt{\kappa(\mathbf{x}_i, \mathbf{x}_i)}$$

- Note that the resulting bound on the number of iterations

$$\frac{R^2}{\gamma^2}$$

has no explicit dependence on the dimension of the feature space.

Proof of Novikoff theorem

- The proof follows from two bounds:

$$\|\mathbf{w}_t\|^2 \leqslant tR^2 \text{ and } \|\mathbf{w}_t\| \geqslant t\gamma$$

Proof of Novikoff theorem

- The proof follows from two bounds:

$$\|\mathbf{w}_t\|^2 \leqslant tR^2 \text{ and } \|\mathbf{w}_t\| \geqslant t\gamma$$

- Assuming these for the moment we have

$$t^2\gamma^2 \leqslant \|\mathbf{w}_t\|^2 \leqslant tR^2 \text{ implying } t \leqslant \frac{R^2}{\gamma^2}$$

Proof of Novikoff theorem

- The proof follows from two bounds:

$$\|\mathbf{w}_t\|^2 \leqslant tR^2 \text{ and } \|\mathbf{w}_t\| \geqslant t\gamma$$

- Assuming these for the moment we have

$$t^2\gamma^2 \leqslant \|\mathbf{w}_t\|^2 \leqslant tR^2 \text{ implying } t \leqslant \frac{R^2}{\gamma^2}$$

- The first inequality follows from assumption of misclassification

$$\|\mathbf{w}_{t+1}\|^2 = \|\mathbf{w}_t + y_j\mathbf{x}_j\|^2 = \|\mathbf{w}_t\|^2 + 2y_j\langle\mathbf{w}_t, \mathbf{x}_j\rangle + \|\mathbf{x}_j\|^2 \leqslant \|\mathbf{w}_t\|^2 + R^2$$

and the second from the existence of $\mathbf{w}$:

$$\|\mathbf{w}_t\| \geqslant \langle\mathbf{w}, \mathbf{w}_t\rangle \geqslant \langle\mathbf{w}, \mathbf{w}_{t-1}\rangle + y_j\langle\mathbf{w}, \mathbf{x}_j\rangle \geqslant t\gamma$$