9. Kernel Methods
Motivation

- Usually learning algorithms assume that some kind of feature function is given
- Reasoning is then done on a feature vector of a given (finite) length
- But: some objects are hard to represent with a fixed-size feature vector, e.g. text documents, molecular structures, evolutionary trees
- Idea: use a way of measuring similarity without the need of features, e.g. the edit distance for strings
- This we will call a kernel function
Dual Representation

Many problems can be expressed using a dual formulation. Example (linear regression):

\[
J(w) = \frac{1}{2} \sum_{n=1}^{N} (w^T \phi(x_n) - t_n)^2 + \frac{\lambda}{2} w^T w \quad \phi(x_n) \in \mathbb{R}^D
\]
Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

\[
J(w) = \frac{1}{2} \sum_{n=1}^{N} (w^T \phi(x_n) - t_n)^2 + \frac{\lambda}{2} w^T w \quad \phi(x_n) \in \mathbb{R}^D
\]

if we write this in vector form, we get

\[
J(w) = \frac{1}{2} w^T \Phi^T \Phi w - w \Phi^T t + \frac{1}{2} t^T t + \frac{\lambda}{2} w^T w \quad t \in \mathbb{R}^N
\]
Dual Representation

Many problems can be expressed using a dual formulation. Example (linear regression):

\[ J(w) = \frac{1}{2} \sum_{n=1}^{N} (w^T \phi(x_n) - t_n)^2 + \frac{\lambda}{2} w^T w \]

\[ \phi(x_n) \in \mathbb{R}^D \]

if we write this in vector form, we get

\[ J(w) = \frac{1}{2} w^T \Phi^T \Phi w - w \Phi^T t + \frac{1}{2} t^T t + \frac{\lambda}{2} w^T w \]

\[ t \in \mathbb{R}^N \]

and the solution is

\[ w = (\Phi^T \Phi + \lambda I_D)^{-1} \Phi^T t \]
Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

\[ J(w) = \frac{1}{2} w^T \Phi^T \Phi w - w \Phi^T t + \frac{1}{2} t^T t + \frac{\lambda}{2} w^T w \]

\[ w = (\Phi^T \Phi + \lambda I_D)^{-1} \Phi^T t \]

However, we can express this result in a different way using the **matrix inversion lemma**:

\[ (A + B C D)^{-1} = A^{-1} - A^{-1} B (C^{-1} + D A^{-1} B)^{-1} D A^{-1} \]
Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

\[
J(w) = \frac{1}{2} w^T \Phi^T \Phi w - w \Phi^T t + \frac{1}{2} t^T t + \frac{\lambda}{2} w^T w
\]

\[
w = (\Phi^T \Phi + \lambda I_N)^{-1} \Phi^T t
\]

However, we can express this result in a different way using the **matrix inversion lemma**:

\[
(A + BCD)^{-1} = A^{-1} - A^{-1} B (C^{-1} + DA^{-1} B)^{-1} D A^{-1}
\]

\[
w = \Phi^T (\Phi \Phi^T + \lambda I_N)^{-1} t
\]
Dual Representation

Many problems can be expressed using a dual formulation. Example (linear regression):

\[ J(w) = \frac{1}{2} w^T \Phi^T \Phi w - w \Phi^T t + \frac{1}{2} t^T t + \frac{\lambda}{2} w^T w \]

\[ w = (\Phi^T \Phi + \lambda I_D)^{-1} \Phi^T t \]

\[ w = \Phi^T (\Phi \Phi^T + \lambda I_N)^{-1} t \]

Plugging \( w = \Phi^T a \) into \( J(w) \) gives:

\[ J(a) = \frac{1}{2} a^T \Phi \Phi^T \Phi \Phi^T a - a^T \Phi \Phi^T t + t^T t + \frac{\lambda}{2} a^T \Phi \Phi^T a \]

“Dual Variables”
Dual Representation

Many problems can be expressed using a dual formulation. Example (linear regression):

\[ J(w) = \frac{1}{2} w^T \Phi^T \Phi w - w \Phi^T t + \frac{1}{2} t^T t + \frac{\lambda}{2} w^T w \]

\[ J(a) = \frac{1}{2} a^T K K a - a^T K t + \frac{1}{2} t^T t + \frac{\lambda}{2} a^T K a \quad K = \Phi \Phi^T \]

This is called the dual formulation.

Note: \[ a \in \mathbb{R}^N \quad w \in \mathbb{R}^D \]
Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

\[
J(w) = \frac{1}{2} w^T \Phi^T \Phi w - w \Phi^T t + \frac{1}{2} t^T t + \frac{\lambda}{2} w^T w
\]

\[
J(a) = \frac{1}{2} a^T KK a - a^T K t + \frac{1}{2} t^T t + \frac{\lambda}{2} a^T K a
\]

This is called the **dual formulation**. The solution to the dual problem is:

\[
a = (K + \lambda I_N)^{-1} t
\]
Many problems can be expressed using a dual formulation. Example (linear regression):

\[
J(w) = \frac{1}{2} w^T \Phi^T \Phi w - w \Phi^T t + \frac{1}{2} t^T t + \frac{\lambda}{2} w^T w
\]

\[
J(a) = \frac{1}{2} a^T K K a - a^T K t + \frac{1}{2} t^T t + \frac{\lambda}{2} a^T K a
\]

\[
a = (K + \lambda I_N)^{-1} t
\]

This we can use to make predictions:

\[
y(x) = w^T \phi(x) = a^T \Phi \phi(x) = k(x)^T (K + \lambda I_N)^{-1} t
\]

(now \(x\) is unknown and \(a\) is given from training)
Dual Representation

\[ y(x) = k(x)^T (K + \lambda I_N)^{-1} t \]

where:

\[
k(x) = \begin{pmatrix}
\phi(x_1)^T \phi(x) \\
\vdots \\
\phi(x_N)^T \phi(x)
\end{pmatrix} \quad K = \begin{pmatrix}
\phi(x_1)^T \phi(x_1) & \cdots & \phi(x_1)^T \phi(x_N) \\
\vdots & \ddots & \vdots \\
\phi(x_N)^T \phi(x_1) & \cdots & \phi(x_N)^T \phi(x_N)
\end{pmatrix}
\]

Thus, \( y \) is expressed only in terms of **dot products** between different pairs of \( \phi(x) \), or in terms of the kernel function

\[
k(x_i, x_j) = \phi(x_i)^T \phi(x_j)
\]
Representation using the Kernel

\[ y(x) = k(x)^T (K + \lambda I_N)^{-1} t \]

Now we have to invert a matrix of size \( N \times N \), before it was \( M \times M \) where \( M < N \), but:

By expressing everything with the kernel function, we can deal with very high-dimensional or even infinite-dimensional feature spaces!

**Idea**: Don’t use features at all but simply define a similarity function expressed as the kernel!
Constructing Kernels

The straightforward way to define a kernel function is to first find a basis function $\phi(x)$ and to define:

$$k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

This means, $k$ is an inner product in some space $\mathcal{H}$, i.e:

1. Symmetry: $k(x_i, x_j) = \langle \phi(x_j), \phi(x_i) \rangle = \langle \phi(x_i), \phi(x_j) \rangle$
2. Linearity: $\langle a(\phi(x_i) + z), \phi(x_j) \rangle = a \langle \phi(x_i), \phi(x_j) \rangle + a \langle z, \phi(x_j) \rangle$
3. Positive definite: $\langle \phi(x_i), \phi(x_i) \rangle \geq 0$, equal if $\phi(x_i) = 0$

Can we find conditions for $k$ under which there is a (possibly infinite dimensional) basis function into $\mathcal{H}$, where $k$ is an inner product?
Constructing Kernels

**Theorem (Mercer):** If $k$ is

1. symmetric, i.e. $k(x_i, x_j) = k(x_j, x_i)$ and
2. positive definite, i.e.

\[
K = \begin{pmatrix}
  k(x_1, x_1) & \ldots & k(x_1, x_N) \\
  \vdots & \ddots & \vdots \\
  k(x_N, x_1) & \ldots & k(x_N, x_N)
\end{pmatrix}
\]

is positive definite, then there exists a mapping $\phi(x)$ into a feature space $\mathcal{H}$ so that $k$ can be expressed as an inner product in $\mathcal{H}$.

**This means, we don’t need to find $\phi(x)$ explicitly!**

**We can directly work with $k$** "Kernel Trick"
Constructing Kernels

Finding valid kernels from scratch is hard, but:
A number of rules exist to create a new valid kernel \( k \) from given kernels \( k_1 \) and \( k_2 \). For example:

\[
\begin{align*}
  k(x_1, x_2) &= ck_1(x_1, x_2), \quad c > 0 \\
  k(x_1, x_2) &= f(x_1)k_1(x_1, x_2)f(x_2) \\
  k(x_1, x_2) &= \exp(k_1(x_1, x_2)) \\
  k(x_1, x_2) &= k_1(x_1, x_2) + k_2(x_1, x_2) \\
  k(x_1, x_2) &= k_1(x_1, x_2)k_2(x_1, x_2) \\
  k(x_1, x_2) &= x_1^T A x_2
\end{align*}
\]

where \( A \) is positive semidefinite and symmetric.
Examples of Valid Kernels

- Polynomial Kernel:
  \[ k(x_i, x_j) = (x_i^T x_j + c)^d \quad c > 0 \quad d \in \mathbb{N} \]

- Gaussian Kernel:
  \[ k(x_i, x_j) = \exp(-\|x_i - x_j\|^2 / 2\sigma^2) \]

- Kernel for sets:
  \[ k(A_1, A_2) = 2^{|A_1 \cap A_2|} \]

- Matern kernel:
  \[ k(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left( \frac{\sqrt{2\nu r}}{l} \right)^\nu K_\nu \left( \frac{\sqrt{2\nu r}}{l} \right) \quad r = \|x_i - x_j\|, \nu > 0, l > 0 \]
A Simple Example

Define a kernel function as

\[ k(x, x') = (x^T x')^2 \quad \text{for} \quad x, x' \in \mathbb{R}^2 \]

This can be written as:

\[
(x_1 x'_1 + x_2 x'_2)^2 = x_1^2 x_1'^2 + 2 x_1 x_1' x_2 x_2' + x_2^2 x_2'^2
\]

\[
= (x_1^2, x_2^2, \sqrt{2} x_1 x_2) (x_1'^2, x_2'^2, \sqrt{2} x_1' x_2')^T
\]

\[
= \phi(x)^T \phi(x')
\]

It can be shown that this holds in general for

\[ k(x_i, x_j) = (x_i^T x_j)^d \]
Visualization of the Example

Original decision boundary is an ellipse

\[
\phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)
\]

Decision boundary becomes a hyperplane
Application Examples

Kernel Methods can be applied for many different problems, e.g.:

- Density estimation (unsupervised learning)
- Regression
- Principal Component Analysis (PCA)
- Classification

Most important Kernel Methods are:

- Support Vector Machines
- Gaussian Processes
Kernelization

• Many existing algorithms can be converted into kernel methods
• This process is called “kernelization”

Idea:
• express similarities of data points in terms of an inner product (dot product)
• replace all occurrences of that inner product by the kernel function

This is called the **kernel trick**
Example: Nearest Neighbor

- The NN classifier selects the label of the nearest neighbor in Euclidean distance

\[ \|x_i - x_j\|^2 = x_i^T x_i + x_j^T x_j - 2x_i^T x_j \]
Example: Nearest Neighbor

- The NN classifier selects the label of the nearest neighbor in Euclidean distance

\[ \| x_i - x_j \|^2 = x_i^T x_i + x_j^T x_j - 2 x_i^T x_j \]

- We can now replace the dot products by a valid Mercer kernel and we obtain:

\[ d(x_i, x_j)^2 = k(x_i, x_i) + k(x_j, x_j) - 2 k(x_i, x_j) \]

- This is a kernelized nearest-neighbor classifier
- We do not explicitly compute feature vectors!
Example: Principal Component Analysis

• Given: data set \( \{x_n\} \quad n = 1, \ldots, N \quad x_n \in \mathbb{R}^D \)

• Project data onto a subspace of dimension \( M \) so that the variance is maximized ("decorrelation")

• For now: assume \( M \) is equal to 1

• Thus: the subspace can be described by a \( D \)-dimensional unit vector \( u_1 \), i.e.: \( u_1^T u_1 = 1 \)

• Each data point is projected onto the subspace using the dot product: \( u_1^T x_n \)
Principal Component Analysis

Visualization:

Mean:

\[ \mu = \frac{1}{N} \sum_{n=1}^{N} u_1^T x_n = \frac{1}{N} u_1^T \sum_{n=1}^{N} x_n = u_1^T \bar{x} \]

Variance:

\[ \sigma^2 = \frac{1}{N} \sum_{n=1}^{N} (u_1^T x_n - u_1^T \bar{x})^2 = \frac{1}{N} \sum_{n=1}^{N} (u_1^T (x_n - \bar{x}))^2 = u_1^T \frac{1}{N} \sum_{n=1}^{N} (x_n - \bar{x})(x_n - \bar{x})^T u_1 \]
Principal Component Analysis

Goal: Maximize $u_1^T S u_1$ s.t. $u_1^T u_1 = 1$

Using a Lagrange multiplier:

$$u^* = \arg \max_{u_1} u_1^T S u_1 + \lambda_1 (1 - u_1^T u_1)$$

Setting the derivative wrt. $u_1$ to 0 we obtain:

$$S u_1 = \lambda_1 u_1$$

Thus: $u_1$ must be an eigenvector of $S$.

Multiplying with $u_1^T$ from left gives: $u_1^T S u_1 = \lambda_1$

Thus: $\sigma^2$ is largest if $u_1$ is the eigenvector of the largest eigenvalue of $S$
Principal Component Analysis

We can continue to find the best one-dimensional subspace that is orthogonal to $u_1$

If we do this $M$ times we obtain:

$u_1, \ldots, u_M$ are the eigenvectors of the $M$ largest eigenvalues of $S$: $\lambda_1, \ldots, \lambda_M$

To project the data onto the $M$-dimensional subspace we use the dot-product:

$$x^\perp = \begin{pmatrix} u_1^T \\ \vdots \\ u_M^T \end{pmatrix} (x - \bar{x})$$
Reconstruction using PCA

• We can interpret the vectors $\mathbf{u}_1, \ldots, \mathbf{u}_M$ as a basis if $M = D$

• A reconstruction of a data point $\mathbf{x}$ into an $M$-dimensional subspace ($M < D$) can be written:

$$\tilde{\mathbf{x}}_n = \sum_{i=1}^{M} z_{ni} \mathbf{u}_i + \sum_{i=M+1}^{D} b_i \mathbf{u}_i$$

• Goal is to minimize the squared error:

$$J = \frac{1}{N} \sum_{n=1}^{N} \left\| \mathbf{x}_n - \tilde{\mathbf{x}}_n \right\|^2$$

• This results in:

$$z_{ni} = \mathbf{x}_n^T \mathbf{u}_i \quad b_i = \bar{\mathbf{x}}^T \mathbf{u}_i$$

These are the coefficients of the eigenvectors
Reconstruction using PCA

Plugging in, we have:

\[
\tilde{x}_n = \sum_{i=1}^{M} (x_n^T u_i) u_i + \sum_{i=M+1}^{D} (\bar{x}^T u_i) u_i \\
= \sum_{i=1}^{D} (\bar{x}^T u_i) u_i - \sum_{i=1}^{M} (\bar{x}^T u_i) u_i + \sum_{i=1}^{M} (x_n^T u_i) u_i \\
= \bar{x} + \sum_{i=1}^{M} (x_n^T u_i - \bar{x}^T u_i) u_i \\
= \bar{x} + \sum_{i=1}^{M} ((x_n - \bar{x})^T u_i) u_i
\]

1. Subtract mean
2. Project onto first \( M \) eigenvectors
3. Back-project
4. Add mean
Application of PCA: Face Recognition

Database

Image to identify

Identification
Application of PCA: Face Recognition

Approach:

• Convert the image into a nm vector by stacking the columns:

• A small image is 100x100 -> a 10000 element vector, i.e. a point in a 10000 dimension space

• Then compute covariance matrix and eigenvectors

• Select number of dimensions in subspace

• Find nearest neighbor in subspace for a new image
Results of Face Recognition

- 30% of faces used for testing, 70% for learning.
Can We Use Kernels in PCA?

• What if data is distributed along non-linear principal components?
• **Idea:** Use non-linear kernel to map into a space where PCA can be done
Kernel PCA

Here, assume that the mean of the data is zero:

\[ \sum_{n=1}^{N} x_n = 0 \]

Then, in standard PCA we have the eigenvalue problem:

\[ Su_i = \lambda_i u_i \quad S = \frac{1}{N} \sum_{n=1}^{N} x_n x_n^T \]

Now, we use a non-linear transformation \( \phi(x_n) \) and we assume \( \sum_{n=1}^{N} \phi(x_n) = 0 \). We define \( C \) as

\[ C = \frac{1}{N} \sum_{n=1}^{N} \phi(x_n)\phi(x_n)^T \hspace{1cm} \text{with} \hspace{1cm} C v_i = \lambda_i v_i \]

Goal: find eigenvalues without using features!
Kernel PCA

Plugging in:
\[
\frac{1}{N} \sum_{n=1}^{N} \phi(x_n) \phi(x_n)^T v_i = \lambda_i v_i
\]

This means, there are values \( a_{in} \) so that \( v_i = \sum_{i=1}^{N} a_{in} \phi(x_n) \). With this we have:
\[
\frac{1}{N} \sum_{n=1}^{N} \phi(x_n) \phi(x_n)^T \sum_{m=1}^{N} a_{im} \phi(x_m) = \lambda_i \sum_{i=1}^{N} a_{in} \phi(x_n)
\]

Multiplying both sides by \( \phi(x_l) \) gives:
\[
\frac{1}{N} \sum_{n=1}^{N} k(x_l, x_n) \sum_{m=1}^{N} a_{im} k(x_n, x_m) = \lambda_i \sum_{i=1}^{N} a_{in} k(x_l, x_n)
\]

where \( k(x_l, x_n) = \phi(x_l)^T \phi(x_n) \). This is our expression in terms of the kernel function!
Kernel PCA

The problem can be cast as finding eigenvectors of the kernel matrix $K$:

$$K a_i = \lambda_i N a_i$$

With this, we can find the projection of the image of $x$ onto a given principal component as:

$$\phi(x)^T v_i = \sum_{n=1}^{N} a_{in} \phi(x)^T \phi(x_n) = \sum_{n=1}^{N} a_{in} k(x, x_n)$$

Again, this is expressed in terms of the kernel function.
Kernel PCA: Example

Eigenvalue=21.72

Eigenvalue=21.65

Eigenvalue=4.11

Eigenvalue=3.93

Eigenvalue=3.66

Eigenvalue=3.09

Eigenvalue=2.60

Eigenvalue=2.53
Example: Classification

• We have seen kernel methods for density estimation, PCA and regression
• For classification there are two major kernel methods: Support Vector Machines (SVMs) and Gaussian Processes
• SVMs are probably the most used classification algorithm
• Main idea: use kernelisation to map into a high-dimensional feature space, where a linear separation between the classes can be found ("hyper-plane")
Support Vector Machines

Support Vector Machines learn a linear discriminant function ("hyper-planes"):

\[ y(x, w) = w^T \phi(x) + b \]

Assumptions for now: Data is linearly separable, Binary classification (\( t_i \in \{-1; +1\} \)).

“Maximum Margin”: find the decision boundary that maximizes the distance to the closest data point
Maximum Margin

\[ \frac{|y(x)|}{\|w\|} \]

Points with minimal distance are called "Support Vectors".

"Support Vectors"

\( w \)

margin

linear decision boundary
Maximum Margin

- The distance of a point \( x_n \) to the decision hyperplane is

\[
\frac{|y(x_n)|}{||w||} = \frac{t_n y(x_n)}{||w||} = \frac{t_n (w^T \phi(x_n) + b)}{||w||}
\]

- This distance is independent of the scale of \( w \) and \( b \)

\[
\frac{t_n (\alpha w^T \phi(x_n) + \alpha b)}{||\alpha w||} = \frac{t_n (w^T \phi(x_n) + b)}{||w||}
\]

- Maximum margin is found by

\[
\arg \max_{w,b} \left\{ \frac{1}{||w||} \min_n \{t_n (w^T \phi(x_n) + b)\} \right\}
\]

- Rescaling: We can choose \( \alpha \) so that

\[
t_n (\alpha w^T \phi(x_n) + \alpha b) = 1
\]
Rescaling

\[
\frac{1}{\|w\|}
\]

\[w\]
Rescaling

\[ \frac{1}{||w||} \]
Maximum Margin

For all data points we have the constraint

\[ t_n(w^T \phi(x_n) + b) \geq 1, \quad n = 1, \ldots, N \]

This means we have to maximize:

\[
\arg \max_{w,b} \left\{ \frac{1}{\|w\|} \right\} \quad \text{s.th.} \quad t_n(w^T \phi(x_n) + b) \geq 1, \quad n = 1, \ldots, N
\]

which is equivalent to

\[
\arg \min_{w,b} \left\{ \frac{1}{2} \|w\|^2 \right\} \quad \text{s.th.} \quad t_n(w^T \phi(x_n) + b) \geq 1, \quad n = 1, \ldots, N
\]
Maximum Margin

\[ \arg \min_{w, b} \left\{ \frac{1}{2} \|w\|^2 \right\} \quad \text{s.th.} \quad t_n(w^T \phi(x_n) + b) \geq 1, \quad n = 1, \ldots, N \]

This is a constrained optimization problem. It can be solved with a technique called quadratic programming.
**Dual Formulation**

For the constrained minimization we can introduce **Lagrange multipliers** $a_n$:

\[
\min L(w, b, a) = \frac{1}{2} \|w\|^2 - \sum_{n=1}^{N} a_n \left( t_n (w^T \phi(x_n) + b) - 1 \right)
\]

Setting the derivatives of this wrt. $w$ and $b$ to 0 yields:

\[
w = \sum_{n=1}^{N} a_n t_n \phi(x_n) \quad \quad 0 = \sum_{n=1}^{N} a_n t_n
\]

If we plug these constraints back into $L(w, b, a)$:

\[
\max \tilde{L}(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(x_n, x_m)
\]
Dual Formulation

\[ \max \tilde{L}(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(x_n, x_m) \]

subject to the constraints

\[ a_n \geq 0, \quad n = 1, \ldots, N \]
\[ \sum_{n=1}^{N} a_n t_n = 0 \]

This is called the \textit{dual formulation} of the constrained optimization problem. The function \( k \) is again the \textit{kernel function} and is defined as:

\[ k(x_n, x_m) = \phi(x_n^T)\phi(x_m) \]

The simplest example of a kernel function is given for \( \Phi = 1 \). It is also known as the \textit{linear kernel}.

\[ k(x_n, x_m) = x_n^T x_m \]
The Kernel Trick in SVMs

- Other kernels are possible, e.g. the polynomial:

\[ \phi(x) = (x_1^2, x_2^2, x_1 x_2, x_2 x_1) \quad x \in \mathbb{R}^2 \]

\[ k(x_n, x_m) = \phi(x_n^T) \phi(x_m) = (x^T x)^2 \]

**Kernel Trick for SVMs:** If we find an optimal solution to the dual form of our constrained optimization problem, then we can replace the kernel by any other valid kernel and obtain again an optimal solution.

- Consequence: Using a non-linear feature transform \( \Phi \) we obtain non-linear decision boundaries.
Observations and Remarks

• The kernel function is evaluated for each pair of training data points during training.

• It can be shown that for every training data point it holds either $a_n = 0$ or $t_n y(x_n) = 1$. In the latter case, they are support vectors.

• For classifying a new feature vector $x$ we evaluate:

$$y(x) = \sum_{n=1}^{N} a_n t_n k(x, x_n) + b$$

We only need to compute that for the support vectors.
Multiple Classes

We can generalize the binary classification problem for the case of multiple classes. This can be done with:

• one-to-many classification
• Defining a single objective function for all classes
• Organizing pairwise classifiers in a directed acyclic graph (DAGSVM)
Extension: Non-separable problems

\[ \frac{|y(x)|}{\|w\|} \]

margin
Slack Variables

- The slack variable $\xi_n$ is defined as follows:
- For all points on the correct side: $\xi_n = 0$
- For all other points: $\xi_n = |t_n - y(x_n)|$
- This means that points with $0 < \xi_n \leq 1$ are correct classified, but inside the margin, points with $\xi_n > 1$ are misclassified.
- In the optimization, we modify the constraints:
  \[ t_n y(x_n) \geq 1 - \xi_n, \quad n = 1, \ldots, N \]
- and $\xi_n \geq 0$
Summary

- Kernel methods are used to solve problems by implicitly mapping the data into a (high-dimensional) feature space.
- The feature function itself is not used, instead the algorithm is expressed in terms of the kernel.
- Applications are manifold, including density estimation, regression, PCA and classification.
- An important class of kernelized classification algorithms are Support Vector Machines.
- They learn a linear discriminative function, which is called a hyper-plane.
- Learning in SVMs can be done efficiently.