KERNEL METHODS
Outline

- Dual representation
- Constructing kernels
- Radial basis functions
- Gaussian processes
In linear **parametric** regression or classification we model target variables $t$ as functions $y(x, w)$ of the input vector $x$, where $w$ is a parameter vector.

1. Estimate $w$ from the training input vectors $x_n$ and their associated target variables $t_n$
2. Throw away the training data $(x_n, t_n)$
3. Given a new input vector $x$, estimate the corresponding target variable $t$, using the learned parameters $w$:

$$t \approx y(x, w)$$
Parametric vs Kernel Methods

- In **kernel** methods we model new data \((x, t)\) directly as a function of the training data \((x_n, t_n)\).
  
  - This means we never throw away the training data.
  
  - The kernel method allows a high-dimensional (even infinite dimensional) feature space to be used implicitly.
The kernel function

The kernel function $k(x, x')$ measures the 'similarity' of input vectors $x$ and $x'$ as an inner product in a feature space defined by the feature space mapping $\phi(x)$:

$$k(x, x') = \phi(x)^t \phi(x')$$

If $k(x, x') = k(x - x')$ we say that the kernel is *stationary*.

If $k(x, x') = k(||x - x'||)$ we call it a *radial basis function*.
Dual Representations

- Many linear models for regression and classification can be reformulated as dual kernel models.

- For example, regularized linear least-squares regression, where the error function is given by:

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

- which has a minimum at

\[ w = -\frac{1}{\lambda} \sum_{n=1}^{N} \left( w^t \phi(x_n) - t_n \right) \phi(x_n) \]
Dual Representations

\[
    \mathbf{w} = -\frac{1}{\lambda} \sum_{n=1}^{N} \{ \mathbf{w}^t \phi(\mathbf{x}_n) - t_n \} \phi(\mathbf{x}_n)
\]

Defining \( a_n = -\frac{1}{\lambda} \{ \mathbf{w}^t \phi(\mathbf{x}_n) - t_n \} \)

we can reexpress \( \mathbf{w} \) as

\[
    \mathbf{w} = \Phi^t \mathbf{a}
\]

where \( \phi(\mathbf{x}_n)^t \) is the \( n^{th} \) row of \( \Phi \)

and \( a_n \) is the \( n^{th} \) element of \( \mathbf{a} \)
Dual Representations

Substituting, we can express the error function in terms of $a$:

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

which has a minimum at

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

where $K$ is the Gram matrix $K = \Phi \Phi^t$ with elements

$$K_{nm} = \phi(x_n)^t \phi(x_m) = k(x_n, x_m)$$
Dual Representations

Predictions are then given by

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

where \( k(x) \) is the vector with elements \( k_n(x) = k(x_n, x) \)

and \( k(x_n, x_m) = \phi(x_n)^t \phi(x_m) \)

- Thus predictions are expressed entirely in terms of kernel functions on input vector pairs.
Constructing Kernels

- One can construct a kernel by selecting a feature space mapping $\phi(x)$ and then defining
  
  $$k(x, x') = \phi(x)^t \phi(x')$$

![Polynomial Kernel](image1)

![Gaussian Kernel](image2)
We can construct a kernel by selecting a feature space mapping $\phi(x)$ and then defining

$$k(x, x') = \phi(x)^t \phi(x')$$
Constructing Kernels

- Alternatively, we can construct the kernel function directly, ensuring that it corresponds to an inner product in some (possibly infinite-dimensional) feature space.
Constructing Kernels

\[ k(x) = \phi(x)^t \phi(x') \]

Example 1: \( k(x, z) = x^t z \)

Example 2: \( k(x, z) = x^t z + c, \ c > 0 \)

Example 3: \( k(x, z) = (x^t z)^2 \)
Constructing Kernels

More generally, $k(x, x')$ is a valid kernel if the Gram matrix $K$ whose elements are given by $k(x_n, x_m)$ is positive semidefinite for all possible choices of $\{x_n\}$.
Kernel Properties

- Kernels obey certain properties that make it easy to construct complex kernels from simpler ones.
Kernel Properties

Given valid kernels $k_1(x, x')$ and $k_2(x, x')$ the following kernels will also be valid:

\[
\begin{align*}
k(x, x') &= ck_1(x, x') \\
k(x, x') &= f(x)k_1(x, x')f(x') \\
k(x, x') &= q(k_1(x, x')) \\
k(x, x') &= \exp(k_1(x, x')) \\
k(x, x') &= k_1(x, x') + k_2(x, x') \\
k(x, x') &= k_1(x, x')k_2(x, x') \\
k(x, x') &= k_3(\phi(x), \phi(x')) \\
k(x, x') &= x^T A x' \\
k(x, x') &= k_a(x_a, x_a') + k_b(x_b, x_b') \\
k(x, x') &= k_a(x_a, x_a')k_b(x_b, x_b')
\end{align*}

where $c > 0$, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\phi(x)$ is a mapping from $\mathbb{R} \to \mathbb{R}^M$, $A$ is a symmetric positive semidefinite matrix, $x_a$ and $x_b$ are variables such that $x = (x_a, x_b)$ and $k_a, k_b$ are valid kernels over their respective spaces.
Constructing Kernels

- Examples:

\[ k(x^t, x') = \left( x^t x' + c \right)^M, c > 0 \] (Use 6.18)

\[ k(x, x') = \exp\left( -\frac{\|x - x'\|^2}{2\sigma^2} \right) \] (Use 6.14 and 6.16.)

Corresponds to infinite-dimensional feature vector
Radial Basis Functions

- Widely used choice of basis function for linear regression.

For example, given training input vectors \( \{ x_1, \ldots, x_N \} \) and corresponding target values \( \{ t_1, \ldots, t_N \} \), we can define an interpolation function \( f(x) \) as:

\[
f(x) = \sum_{n=1}^{N} w_n h(\|x - x_n\|)
\]

Since \( w \) and \( t \) have the same dimensionality, we can fit the training data exactly, while providing predictions for input vectors we haven't yet seen.
Kernel Regression

- Can also be used for prediction when there is noise in $\mathbf{x}$, as well as $t$ (the **Nadaraya-Watson** model):

Let $\xi$ be the noise in $\mathbf{x}$, drawn from distribution $\nu(\xi)$. Then

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

Using the calculus of variations (Appendix D), we can find the optimal interpolation function $y(\mathbf{x})$:

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

where

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

\[ y(x) = \sum_{n=1}^{N} t_n h(x - x_n), \text{ where } h(x - x_n) = \frac{v(x - x_n)}{\sum_{n=1}^{N} v(x - x_n)} \]

\[ v(x - x_n) \] and \[ h(x - x_n) \] are shown in the diagrams.
The Nadaraya-Watson model can also be understood as a form of kernel density estimation:

Given a training set \( \{ x_n, t_n \} \), we model the joint distribution \( p(x,t) \) using a Parzen density estimator

\[
p(x,t) = \frac{1}{N} \sum_{n=1}^{N} f(x - x_n, t - t_n)
\]

Then the regression function \( y(x) \) is given by

\[
y(x) = E[t | x] = \sum_{n} k(x, x_n) t_n
\]

where \( k(x, x_n) = \frac{g(x - x_n)}{\sum_{m} g(x - x_m)} \) and \( g(x) = \int_{-\infty}^{\infty} f(x, t) dt \)
The model provides not only the expected target value, but the entire posterior:

Given a training set \( \{x_n, t_n\} \), we model the joint distribution \( p(x, t) \) using a Parzen density estimator

\[
p(t|\mathbf{x}) = \frac{p(t, \mathbf{x})}{\int p(t, \mathbf{x}) \, dt} = \frac{\sum_n f(x - x_n, t - t_n)}{\sum_m \int f(x - x_m, t - t_m) \, dt}
\]
Gaussian Processes

- In linear regression, we estimate a posterior distribution over parameters $w$.
- This determines a prior distribution $p(y)$ over the function $y$.

$E[t \mid t, \alpha, \beta]$  $p(t \mid t, \alpha, \beta)$

Samples of $y(x,w) = w^t \phi(x)$, where $w$ is sampled from $p(w \mid t)$.
Gaussian Processes

- In the Gaussian process approach, we dispense with the parameters $\mathbf{w}$ and define the prior over $y$ directly.

  More precisely, a Gaussian process defines a probability distribution over functions $y(\mathbf{x})$ such that the set of values of $y(\mathbf{x})$ evaluated at an arbitrary set of points $\mathbf{x}_1, \ldots, \mathbf{x}_N$ are jointly Gaussian.
Equivalence with Classical Regression

\[ y(x) = w^t \phi(x) \]

Assume a Gaussian prior over the weight vector: \( p(w) = N(w \mid 0, \alpha^{-1}I) \).

We have training data \( x_1, \ldots, x_N \).

For given \( w \), this determines a vector of function values \( y = (y(x_1), \ldots, y(x_N))^t \):

\[ y = \Phi w, \quad \text{where} \quad \Phi_{nk} = \phi_k(x_n) \]

Since \( y \) is a linear function of \( w \), it is also Gaussian, with:

\[
E[y] = 0 \\
\text{cov}[y] = \frac{1}{\alpha} \Phi \Phi^t = K,
\]

where \( K_{nm} = k(x_n, x_m) = \frac{1}{\alpha} \phi(x_n) \phi(x_m) \).
Gaussian Processes

- Thus we have a direct probabilistic model for \( y(x_n) \):

\[
p(y) = N(y \mid 0, K)
\]

where \( K_{nm} = k(x_n, x_m) = \frac{1}{\alpha} \phi(x_n) \phi(x_m) \)

- Notes
  - All we need to specify this model is the kernel function evaluated over all pairs of training input vectors.
  - The kernel function can be defined in terms of feature vectors, as above, or directly.
  - Normally, the kernel function is defined so that \( y(x_n) \) and \( y(x_m) \) are strongly correlated if \( x_n \) and \( x_m \) are close to each other.
Gaussian Processes: Examples

Gaussian Kernel

Exponential Kernel
To use Gaussian processes for prediction, we need to account for the noise in the observed target values:

\[ t_n = y_n + \epsilon_n \]

where \( \epsilon_n \) is Gaussian iid, so that:

\[ p(t|y) = N(t|y, \beta^{-1}I) \]
Knowing \( p(y) \) and \( p(t | y) \), we can use the results from Chapter 2 to determine \( p(t) \):

\[
p(t) = N(t | 0, C)
\]

where \( C_{nm} = k(x_n, x_m) + \beta^{-1} \delta_{nm} \)

Predictions can now be made using the partitioned Gaussian equations from Chapter 2.3.2:

\[
p(t_{N+1} | t) = N(t_{N+1} | m(x_{N+1}), \sigma^2(x_{N+1}))
\]

where

\[
m(x_{N+1}) = k^t C_N^{-1} t
\]

\[
k_n = k(x_n, x_{N+1})
\]

\[
\sigma^2(x_{N+1}) = c - k^t C_N^{-1} k
\]

\[
c = k(x_{N+1}, x_{N+1}) + \beta^{-1}
\]
Learning the Hyperparameters

We have assumed that the kernel functions $k(x_n, x_m)$ are fixed. This corresponds to a fixed feature space mapping $\phi(x)$.

In practice, we may instead wish to define the covariance function through a parameterized family of kernels, and then infer the parameters $\theta$ from the data.

In general, this will require iterative optimization and approximation methods.
Gaussian Processes for Classification

For Gaussian process regression, $y$ is Gaussian, and assumes values on $\mathbb{R}$.

For binary classification, where $t \in \{0, 1\}$, we let $a(x)$ be a Gaussian process, and define $y(x) = \sigma(a)$, where $\sigma(a)$ is the sigmoid function

$$
\frac{1}{1 + \exp(-a)}
$$

\[ a(x) \]

\[ \sigma(a(x)) \]
Gaussian Processes for Classification

Now $p(t \mid a) = \sigma(a)^t (1 - \sigma(a))^{1-t}$

Given training data $x_1, \ldots, x_N$, define

$t_N = (t_1, \ldots, t_N)^t$

$a_{N+1} = (a(x_1), \ldots, a(x_{N+1}))^t$

where $p(a_{N+1}) = N(a_{N+1} \mid 0, C_{N+1})$

and $C_{nm} = k(x_n, x_m) + \nu \delta_{nm}$

Then $p(t_{N+1} = 1 \mid t_N) = \int p(t_{N+1} = 1 \mid a_{N+1}) p(a_{N+1} \mid t_N) da_{N+1} = \int \sigma(a_{N+1}) p(a_{N+1} \mid t_N) da_{N+1}$

This integrable is intractable - must approximate. See text for details.
Example

\[ p(t_{N+1} = 1 | t_N) \]