Further Schedule

- Kernels and Gaussian Processes (today)
- Metric Learning
- Clustering, Expectation-Maximization, GMM
- NN and Deep Learning
- Bagging and Boosting
- Probabilistic Graphical Models
- Sequential Data (HMM, CRF)
- Repetition and Applications (Dez. 23)
- Sampling
- Variational Inference
- Online Learning
4. 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):

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

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

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

if we write this in vector form, we get

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

$$\Phi \in \mathbb{R}^{N \times M}$$
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

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

$$t \in \mathbb{R}^N$$

$$\Phi \in \mathbb{R}^{N \times D}$$

and the solution is

$$w = (\Phi^T \Phi + \lambda I_M)^{-1} \Phi^T t$$
Dual Representation

Many problems can be expressed using a **dual** formulation, including linear regression.

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

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

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

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

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

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

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

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

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

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

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

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

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

\[
=: a
\]

“Dual Variables”
Dual Representation

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

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

$$w = (\Phi^T \Phi + \lambda I_M)^{-1} \Phi^T t$$

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

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"
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: \( a \in \mathbb{R}^N \quad w \in \mathbb{R}^M \)
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:

\[ k(x_1, x_2) = ck_1(x_1, x_2), \quad c > 0 \]
\[ k(x_1, x_2) = f(x_1)k_1(x_1, x_2)f(x_2) \]
\[ k(x_1, x_2) = \exp(k_1(x_1, x_2)) \]
\[ k(x_1, x_2) = k_1(x_1, x_2) + k_2(x_1, x_2) \]
\[ k(x_1, x_2) = k_1(x_1, x_2)k_2(x_1, x_2) \]
\[ k(x_1, x_2) = x_1^T A x_2 \]

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

- 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

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

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

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

Original decision boundary is an ellipse

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

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

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

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

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

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

- This is a **kernelized** nearest-neighbor classifier
- We do not explicitly compute feature vectors!
Back to Linear Regression

We had the primal and the dual formulation:

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

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

with the dual solution:

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

This we can use to make predictions (MAP):

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

- We have found a way to predict function values of $y$ for new input points $x$.
- As we used regularized regression, we can equivalently find the **predictive distribution** by marginalizing out the parameters $w$.

Questions:

- Can we find a closed form for that distribution?
- How can we model the uncertainty of our prediction?
- Can we use that for classification?
Gaussian Marginals and Conditionals

First, we need some formulae:

Assume we have two variables $x_a$ and $x_b$ that are **jointly** Gaussian distributed, i.e. $\mathcal{N}(x \mid \mu, \Sigma)$ with

$$
\begin{align*}
    x &= \begin{pmatrix} x_a \\ x_b \end{pmatrix} \\
    \mu &= \begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix} \\
    \Sigma &= \begin{pmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{pmatrix}
\end{align*}
$$

Then the cond. distribution

$$
p(x_a \mid x_b) = \mathcal{N}(x \mid \mu_{a|b}, \Sigma_{a|b})
$$

where

$$
\mu_{a|b} = \mu_a + \Sigma_{ab}\Sigma_{bb}^{-1}(x_b - \mu_b)
$$

and

$$
\Sigma_{a|b} = \Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba} \\
\text{“Schur Complement”}
$$

The marginal is

$$
p(x_a) = \mathcal{N}(x_a \mid \mu_a, \Sigma_{aa})
$$
Gaussian Marginals and Conditionals

Main idea of the proof for the conditional (using inverse of block matrices):

\[
\begin{pmatrix}
\Sigma_{aa} & \Sigma_{ab} \\
\Sigma_{ba} & \Sigma_{bb}
\end{pmatrix}^{-1} = \begin{pmatrix}
I & 0 \\
-\Sigma_{bb}^{-1}\Sigma_{ba} & I
\end{pmatrix} \begin{pmatrix}
(\Sigma/\Sigma_{bb})^{-1} & 0 \\
0 & \Sigma_{bb}^{-1}
\end{pmatrix} \begin{pmatrix}
I & -\Sigma_{ab}\Sigma_{bb}^{-1} \\
0 & I
\end{pmatrix}
\]

The lower line corresponds to a quadratic form that is only dependent on \( p(x_b) \), i.e. the rest can be identified with the conditional Normal distribution \( p(x_a \mid x_b) \).

(for details see, e.g. Bishop or Murphy)
Definition

Definition: A **Gaussian process** is a collection of random variables, any finite number of which have a joint Gaussian distribution.

The number of random variables can be infinite! This means: a GP is a Gaussian distribution over **functions**!

To specify a GP we need:

- mean function: \( m(x) = \mathbb{E}[y(x)] \)
- covariance function:

\[
k(x_1, x_2) = \mathbb{E}[y(x_1) - m(x_1)y(x_2) - m(x_2)]
\]
Example

- green line: sinusoidal data source
- blue circles: data points with Gaussian noise
- red line: mean function of the Gaussian process
How Can We Handle Infinity?

Idea: split the (infinite) number of random variables into a finite and an infinite subset.

\[
\mathbf{x} = \begin{pmatrix} \mathbf{x}_f \\ \mathbf{x}_i \end{pmatrix} \sim \mathcal{N} \left( \begin{pmatrix} \mathbf{\mu}_f \\ \mathbf{\mu}_i \end{pmatrix}, \begin{pmatrix} \sum_f & \sum_{fi} \\ \sum_{fi}^T & \sum_i \end{pmatrix} \right)
\]

From the marginalization property we get:

\[
p(\mathbf{x}_f) = \int p(\mathbf{x}_f, \mathbf{x}_i) d\mathbf{x}_i = \mathcal{N}(\mathbf{x}_f \mid \mathbf{\mu}_f, \Sigma_f)
\]

This means we can use finite vectors.
A Simple Example

In Bayesian linear regression, we had $y(x) = \phi(x)^T w$ with prior probability $w \sim \mathcal{N}(0, \Sigma_p)$. This means:

$$E[y(x)] = \phi(x)^T E[w] = 0$$

$$E[y(x_1)y(x_2))] = \phi(x_1)^T E[ww^T]\phi(x_2) = \phi(x_1)^T \Sigma_p \phi(x_2)$$

Any number of function values $y(x_1), \ldots, y(x_N)$ is jointly Gaussian with zero mean. The covariance function of this process is

$$k(x_1, x_2) = \phi(x_1)^T \Sigma_p \phi(x_2)$$

In general, any valid kernel function can be used.
The Covariance Function

The most used covariance function (kernel) is:

\[ k(x_p, x_q) = \sigma_f^2 \exp\left(-\frac{1}{2l^2}(x_p - x_q)^2\right) + \sigma_n^2 \delta_{pq} \]

signal variance  length scale  noise variance

It is known as “squared exponential”, “radial basis function” or “Gaussian kernel”.

Other possibilities exist, e.g. the exponential kernel:

\[ k(x_p, x_q) = \exp(-\theta|x_p - x_q|) \]

This is used in the “Ornstein-Uhlenbeck” process.
Sampling from a GP

Just as we can sample from a Gaussian distribution, we can also generate samples from a GP. **Every sample will then be a function!**

Process:
1. Choose a number of input points $x_1^*, \ldots, x_M^*$
2. Compute the covariance matrix $K$ where
   \[ K_{ij} = k(x_i^*, x_j^*) \]
3. Generate a random Gaussian vector from
   \[ y_* \sim \mathcal{N}(0, K) \]
4. Plot the values $x_1^*, \ldots, x_M^*$ versus $y_1^*, \ldots, y_M^*$
Sampling from a GP

Squared exponential kernel

Exponential kernel
Prediction with a Gaussian Process

Most often we are more interested in predicting new function values for given input data.

We have:

- training data \( x_1, \ldots, x_N \), \( y_1, \ldots, y_N \)
- test input \( x^*_1, \ldots, x^*_M \)

And we want test outputs \( y^*_1, \ldots, y^*_M \)

The joint probability is

\[
\begin{pmatrix}
  y \\
  y^*
\end{pmatrix}
\sim \mathcal{N}
\begin{pmatrix}
  0, \\
  K(X^*, X)
\end{pmatrix}
\]

\[
\begin{pmatrix}
  K(X, X) & K(X, X^*) \\
  K(X^*, X) & K(X^*, X^*)
\end{pmatrix}
\]

and we need to compute

\[ p(y^* \mid x^*, X, y). \]
Prediction with a Gaussian Process

In the case of only one test point \( x^* \) we have

\[
K(X, x^*) = \begin{pmatrix}
  k(x_1, x^*) \\
  \vdots \\
  k(x_N, x^*)
\end{pmatrix} = k_*
\]

Now we compute the conditional distribution

\[
p(y^* \mid x^*, X, y) = \mathcal{N}(y_* \mid \mu_*, \Sigma_*)
\]

where

\[
\mu_* = k_*^T K^{-1} t
\]

\[
\Sigma_* = k(x_*, x_*) - k_*^T K^{-1} k_*
\]

This defines the predictive distribution.
Example

Functions sampled from a Gaussian Process prior

Functions sampled from the predictive distribution

The predictive distribution is itself a Gaussian process. It represents the posterior after observing the data. The covariance is low in the vicinity of data points.
Varying the Hyperparameters

\( l = \sigma_f = 1, \quad \sigma_n = 0.1 \)

- 20 data samples
- GP prediction with different kernel hyper parameters

\( l = 0.3, \quad \sigma_f = 1.08, \quad \sigma_n = 0.0005 \)

\( l = 3, \quad \sigma_f = 1.16, \quad \sigma_n = 0.89 \)
Algorithm 1: GP regression

Data: training data \((X, y)\), test data \(x_*\)

Input: Hyper parameters \(\sigma_f^2, l, \sigma_n^2\)

\[
K_{ij} \leftarrow k(x_i, x_j) \\
L \leftarrow \text{cholesky}(K + \sigma_n^2 I) \\
\alpha \leftarrow L^T \backslash (L \backslash y) \\
\mathbb{E}[f_*] \leftarrow k^T \alpha \\
v \leftarrow L \backslash k_* \\
\text{var}[f_*] \leftarrow k(x_*, x_*) - v^T v \\
\log p(y \mid X) \leftarrow -\frac{1}{2} y^T \alpha - \sum_i \log L_{ii} - \frac{N}{2} \log(2\pi)
\]

• Cholesky decomposition is numerically stable
• Can be used to compute inverse efficiently
Summary

- Kernel methods 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.
- Gaussian Processes are Normal distributions over functions.
- To specify a GP we need a covariance function (kernel) and a mean function.
- More on Gaussian Processes: http://videolectures.net/epsrcws08_rasmussen_lgp/