Introduction to Kernel Methods

Maschinelles Lernen A
Michael Pfeiffer
pfeiffer@igi.tugraz.at

Plan for Today

- Definition of Kernels
- Motivation from Statistical Learning Theory
- Mathematics of Kernel Methods
- Next Week:
  - Kernel Algorithms: SVM, SV Regression, Kernel PCA
- Talk builds upon:
  - B. Schölkopf: Introduction to Kernel Methods (Erice, 2005)
  - Source of most slides
  - Schölkopf & Smola: Learning with Kernels (Book)

Learning and Similarity

- input/output sets $X, Y$
- training set $(x_1, y_1), \ldots, (x_m, y_m) \in X \times Y$
- "generalization": given a previously unseen $x \in X$, find a suitable $y \in Y$
- $(x, y)$ should be "similar" to $(x_1, y_1), \ldots, (x_m, y_m)$
- how to measure similarity?
  - for outputs: loss function (e.g., for $Y = \{\pm 1\}$, zero-one loss)
  - for inputs: kernel
Similarity of Inputs

- symmetric function
  - \( \langle x, x' \rangle \to \mathbb{R} \)
  - \( \langle x, x' \rangle = \langle x, x' \rangle \)
- for example, if \( X = \mathbb{R}^N \) canonical dot product
  \( \langle x, x' \rangle = \sum_i x_i x'_i \)
- if \( X \) is not a dot product space: assume that it has a representation as a dot product in a linear space \( H \)
  - i.e., there exists a map \( \Phi: X \to H \) such that
    \( \langle x, x' \rangle = \langle \Phi(x), \Phi(x') \rangle \)
- in that case, we can think of the patterns as \( \Phi(x) \), \( \Phi(x') \), and carry out geometric algorithms in the dot product space ("feature space") \( H \).

Example of a Kernel Algorithm

Idea: classify points \( x := \Phi(x) \) in feature space according to which of the two class means is closer.

\[
\begin{align*}
    c_+ &= \frac{1}{m_+} \sum_{i=1}^{m_+} \Phi(x_i), \\
    c_- &= \frac{1}{m_-} \sum_{i=1}^{m_-} \Phi(x_i)
\end{align*}
\]

Compute sign of dot product between \( w := c_+ - c_-, \) and \( x - c \)
- Corresponds to angle

Example of a Kernel Algorithm

\[
\begin{align*}
    f(x) &= \text{sign}(\langle \Phi(x), c_+ - c_- \rangle) \\
         &= \text{sign}(\langle \Phi(x), c_+ + c_- \rangle) \\
         &= \text{sign}(\langle \Phi(x), c \rangle) \\
\end{align*}
\]

\[
\begin{align*}
    b &= \frac{1}{2} \left( \langle c_+, c_+ \rangle - \langle c_-, c_- \rangle \right) \\
    \phi(x) &= \text{sign} \left( \frac{1}{m_+} \sum_{i=1}^{m_+} \langle \Phi(x_i), c \rangle - \frac{1}{m_-} \sum_{i=1}^{m_-} \langle \Phi(x_i), c \rangle \right) \\
\end{align*}
\]

Decision function is hyperplane in feature space
Relation to Parzen Windows

- Assume $b=0$ and $\forall x$
  
  - E.g. RBF kernel
- Parzen window classifiers have kernels centered on training points
  
  - Generalization of k-NN
- Assign new point to class with highest posterior probability
  
  $$p_c(x) = \frac{1}{m} \sum_{i=1}^{m} k(x, x_i)$$

General Kernel Algorithms

- General Form for kernel classification:
  
  $$f(x) = \text{sgn} \left( \sum_{i=1}^{m} \alpha_i \cdot k(x, x_i) + b \right)$$

- Questions:
  - What is the benefit of feature mapping $\Phi$?
  - Relationship between kernel and feature mapping
  - Theoretical guarantees for algorithms

Example: All degree 2 monomials

- Nonlinear mapping $\mathbb{R}^2 \rightarrow \mathbb{R}^3$
- Classification in 3D can be done with a hyperplane
Hyperplane Classifiers

- Hyperplanes can be completely formulated as dot products:
  \[ \langle w, x \rangle + b = 0 \]
- Decision function:
  \[ f(x) = \text{sgn} (\langle w, x \rangle + b) \]
- One of the simplest possible classifiers

Optimal Hyperplanes

- Maximize margin of separation
- Creates robust classifier

Motivation from Statistical Learning Theory

- Set of decision functions \( f: X \rightarrow \{-1, 1\} \)
- Cost function \( c(x, y, f(x)) \), e.g., \( c(x, y, f(x)) = |f(x) - y| \)
- Underlying distribution \( P(x, y) \)
- We want to minimize risk
  \[ R(f) = \int_{X \times Y} c(x, y, f(x)) dP(x, y) \]
- Training error = Empirical Risk
  \[ R_{emp}(f) = \frac{1}{m} \sum_{i=1}^{m} c(x_i, y_i, f(x_i)) \]
- Without restrictions to function class, we cannot control the risk only by minimizing the empirical risk
**VC - Dimension**

- **Definition**: largest $m$, s.t. there exists a set of $m$ points that the function class can shatter.

- Bound on difference between $R$ and $R_{emp}$ for VC-dim $h < m$, independent of distribution $P$:
  
  With prob. $1-\delta$ for drawing $m$ training points:
  
  $$R[f] \leq R_{emp}[f] + \sqrt{\frac{h}{m} \left( \ln \frac{2m}{h} + 1 \right) + \ln \frac{4}{\delta}}$$

**VC-Dimension**

- VC-Dimension tells us the capacity of a class of decision functions.
- Classifiers with high VC-dimension will have small training error, but cannot guarantee low test error.
- Tradeoff between bias and variance of classifier:
  - Small enough to guarantee generalization
  - Large enough to model dependencies

**VC-Dimension and Margin**

- Hyperplanes in $\mathbb{R}^n$ have VC-dimension $h = n+1$.

- Theorem by Vapnik:
  
  By controlling the norm of the weight vector of the separating hyperplanes (i.e. controlling / increasing the margin), we can control the VC-dimension of the function class, irrespective of the dimension of the space.
Summary of Part One

- Kernels compute dot products in high dimensional spaces ~ similarity of inputs
- Hyperplane classifiers can be computed only with dot products
- We can solve non-linearly-separable problems by projecting non-linearly into higher-dimensional spaces and using linear classification there
- By maximizing the margin of separating hyperplanes in high-dimensional feature space, we get low VC-dimension, irrespective of the feature space dimension
- Still open questions:
  - How to compute high-dimensional dot products with kernels?
  - How to select valid kernels?

Mathematics of Kernel Methods

- The “Kernel – Trick”
- Classes of valid kernel functions
- Properties of kernels

Monomial Features

- Consider feature map $\Phi_d(x)$ which computes all possible $d^{th}$ degree products of the entries in $x \in \mathbb{R}^n$
- Dimension of feature space $H$ grows like $N^d$
- e.g. $N = 16 \times 16, d = 5 \Rightarrow \dim(H) \sim 10^{10}$
- e.g. for rather small images, it is almost impossible to compute feature map
- Can we compute dot product between mapped vectors?
We only need to compute dot products in the input space. With this kernel, higher order statistics can be taken into account without combinatorial explosion.

Given some kernel (in the input space), can we construct a feature space such that the kernel computes the dot product in that feature space?

- Does this work even if we have no dot product in the input space?

Any algorithm that depends only on dot products can benefit from the kernel trick.

- We can then apply linear methods to non-vectorial data (e.g. text).
- Kernels are nonlinear similarity measures.
Positive Definite Kernels

- If \( k : X^2 \rightarrow \mathbb{R} \) is symmetric (i.e. \( k(x, x') = k(x', x) \)) and satisfies for any \( m \in \mathbb{N} \) and any set of training points \( x_1, \ldots, x_m \in X \) and any \( a_1, \ldots, a_m \in \mathbb{R} \) the inequality
  \[ \sum_{i,j} a_i a_j K_{ij} \geq 0 \]
  \[ K_{ij} = k(x_i, x_j) \]
  it is called a positive (semi-)definite (p.d.) kernel.

- The matrix \( K = (K_{ij}) \) is called the Gram or kernel matrix of \( k \) with respect to \( x_1, \ldots, x_m \).

Positive Definite Kernels

- Positive Definiteness is defined via the Gram matrix \( K \)

  - Properties of the Gram matrix:
    - A matrix is p.d. if and only if all eigenvalues are nonnegative.
    - Positivity on the Diagonal: \( k(x, x) \geq 0 \) \( \forall x \in X \)
    - Symmetry: \( K = K^T \)
    - Cauchy-Schwarz Inequality: \( |K_{ij}|^2 \leq K_{ii} K_{jj} \) \( \forall i, j \in 1, \ldots, m \)

- If \( \Phi \) maps \( X \) into a dot product space \( H \), then
  \[ k(x, x') = \langle \Phi(x), \Phi(x') \rangle \]
  is a p.d. kernel

- The term kernel comes from integral operators
  \[ (Tf)(x) = \int k(x, x') f(x') \, dx' \]

Reproducing Kernel Map

- For p.d. kernel \( k \) and nonempty set \( X \)
  - Define feature map \( \Phi : X \rightarrow \mathbb{R}^d = T : X \rightarrow \mathbb{R} \)
    \[ \Phi(x)(x) = k(x, x) \]
    - Patterns are turned into functions
  - How to construct feature space:
    1. Turn image of \( \Phi \) into a vector space
    2. Define a dot product that satisfies \( \langle \Phi(x), \Phi(x') \rangle = k(x, x') \)
    3. Complete the space to get a reproducing kernel Hilbert space
Definitions

- Dot product on vector space H:
  - Symmetric, strictly p.d. bilinear form $\langle \cdot, \cdot \rangle: H \times H \to \mathbb{R}$
    - $\langle x, y \rangle = \langle y, x \rangle$
    - $\langle lx + ky, x' \rangle = l\langle x, x' \rangle + k\langle y, x' \rangle$
    - $\langle x, x \rangle > 0$ and $\langle x, x \rangle = 0$ only for $x = 0$

- Dot product space: vector space endowed with dot product
- Hilbert space: A complete dot product space
  - Complete: all Cauchy sequences in the space converge
  - Cauchy sequence: $\forall \varepsilon > 0 \exists N \in \mathbb{N} : \forall n, m > N : \|x_n - x_m\| < \varepsilon$
    - Converges to $x$ if $\|x_n - x\| \to 0$ as $n \to \infty$
    - $\|x\| = \sqrt{\langle x, x \rangle}$

Create a Vector Space

- Form linear combinations
  - $f(.) = \sum_{i=1}^{m} \alpha_i k(\cdot, x_i)$
  - $g(.) = \sum_{j=1}^{m'} \beta_j k(\cdot, x'_j)$
  - $m, m' \in \mathbb{N}$, $\alpha_i, \beta_j \in \mathbb{R}$, $x_i, \ldots, x_m, x'_j, \ldots, x'_{m'} \in X$
- These elements form a vector space

Define Dot Product

- $\langle f, g \rangle := \sum_{i=1}^{m} \sum_{j=1}^{m'} \alpha_i \beta_j \langle x_i, x'_j \rangle$
  - $= \sum_{j=1}^{m'} \beta_j \langle f(\cdot), x'_j \rangle$
- well-defined, although coefficients need not be unique
- Symmetric
- Bilinear
- Is it positive definite, i.e. $\langle f, f \rangle \geq 0$?
Dot Product

- $\langle f, f \rangle = \sum_{i,j} a_i a_j k(x_i, x_j) \geq 0$ because $k$ is p.d.

- $\sum_{j} \gamma_j \langle f_j, f \rangle = \left( \sum_{j} \gamma_j \sum_{j} f_j f_j \right) = \langle f, f \rangle \geq 0$

  for any functions $f_1, \ldots, f_n \in H$ and coefficients $\gamma_1, \ldots, \gamma_n \in \mathbb{R}$

- Thus $\langle \cdot, \cdot \rangle$ is itself a p.d. kernel on the function space $H$

Reproducing Kernels

- $\langle k(\cdot, x), f \rangle = f(x)$ ... $k$ is representer of evaluation
- $\langle k(\cdot, x), k(\cdot, x') \rangle = k(x, x')$ ... reproducing kernel property
- $\Rightarrow k(x, x') = \langle \Phi(x), \Phi(x') \rangle$

- $\|f(x)\|^2 = |\langle k(\cdot, x), f \rangle| \leq k(x, x) \cdot \langle f, f \rangle$
  
  therefore $\langle f, f \rangle = 0$ implies $f = 0$

- Last step to show that $\langle \cdot, \cdot \rangle$ is a dot product

Summary

- We have found one possible feature mapping such that any p.d. kernel corresponds to a dot product in feature space.
- Equivalently, any feature mapping to a dot product space defines a p.d. kernel $k(x, x') = \langle \Phi(x), \Phi(x') \rangle$

- Kernel Trick: Given an algorithm which is formulated in terms of a p.d. kernel $k$, one can construct an alternative algorithm by replacing $k$ by another p.d. kernel $k^*$.
Reproducing Kernel Hilbert Spaces

- A RKHS is a Hilbert space of functions $f$, where all point evaluation functionals $p_x : H \to \mathbb{R}$ exist and are continuous (i.e. when $f$ and $f'$ are close in $H$, $f(x)$ and $f'(x)$ are close in $\mathbb{R}$).
- In that case for each $\mathbf{x} \in X$ there exists a unique function of $x$, called $k(\mathbf{x}, \mathbf{x}')$ s.t.
  \[ f(x') = (f, k(\cdot, x')) \]

Mercer Kernel Map

- We show an alternative mapping to a feature (Hilbert) space: Mercer Kernel Map
  - Traditional way to introduce the kernel trick
- Any two separable Hilbert spaces are isometrically isomorphic:
  - One-to-one linear map between spaces exists, which preserves dot product
- Mercer theorem provides insight into geometry of feature spaces

Mercer’s Theorem

If $k$ is a continuous kernel of a positive definite integral operator on $L^2(X)$ (where $X$ is some compact space),
\[ \int_0^1 k(x, x')f(x)f(x') \, dx \, dx' \geq 0, \]
it can be expanded as
\[ k(x, x') = \sum_{i=1}^\infty \lambda_i \varphi_i(x)\varphi_i(x') \]
using eigenfunctions $\varphi_i$ and eigenvalues $\lambda_i \geq 0$.
Mercer Feature Map

In that case:
\[ \phi(x) = \left( \frac{\sqrt{\lambda_1}(x)}{\lambda_1}, \ldots, \frac{\sqrt{\lambda_n}(x)}{\lambda_n} \right) \]

such that \( \langle \phi(x), \phi(x') \rangle = k(x, x') \).

Proof:
\[ \langle \phi(x), \phi(x') \rangle = \left( \frac{\sqrt{\lambda_1}(x)}{\lambda_1}, \ldots, \frac{\sqrt{\lambda_n}(x)}{\lambda_n} \right) \left( \frac{\sqrt{\lambda_1}(x')}{\lambda_1}, \ldots, \frac{\sqrt{\lambda_n}(x')}{\lambda_n} \right) = \sum_{j=1}^{n} \lambda_j \phi_j(x) \phi_j(x') = k(x, x') \]

Relationship between Reproducing and Mercer Kernels

- Mercer kernels are positive definite kernels
- Thus they are also reproducing kernels
- For a Mercer kernel \( k \) we construct a dot product s.t.
  \( k \) is a reproducing kernel for the Hilbert space of functions \( f \) with

\[
\begin{align*}
f(x) &= \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} a_{mn} \psi_m(\sqrt{\lambda_j}x) y_n(\sqrt{\lambda_j}x) \\
\langle f, \lambda_{mn} \psi_m(\sqrt{\lambda_j}x') y_n(\sqrt{\lambda_j}x') \rangle &= \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} \lambda_j a_{mn} \psi_m(x) y_n(x) \\
&= \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} \psi_m(x') y_n(x') f(x') \\
&= \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} \delta_{m,n} a_{mn} \\
&= \frac{1}{2} \sum_{j=1}^{\infty} \delta_{m,n} \\
&= \begin{cases} 
1 & j = k \\
0 & \text{else}
\end{cases}
\end{align*}
\]

Feature Spaces and Kernels

- Different feature spaces can be constructed for the same kernel
- If \( k(x, x') = \langle \phi_1(x), \phi_1(x') \rangle = \langle \phi_2(x), \phi_2(x') \rangle \)
  - Usually \( \phi_1(x) = \phi_2(x) \)
  - However \( \langle \phi_1(x), \phi_1(x') \rangle = \langle \phi_2(x), \phi_2(x') \rangle \)
- As long as only dot products are considered, the spaces can be regarded as identical
  - Practically we never make use of RKHS or Mercer maps, but only deal with kernel functions
The Empirical Kernel Map

- Remember feature map $\Phi(x) \rightarrow k(.x)$
- Each point is represented by similarity to all other points

- If we have finite training set $x_1, \ldots, x_m$, the kernel $k$ is only evaluated on training patterns
- For linear algorithms, everything takes place in the linear span of mapped training patterns

- Empirical Kernel Map:
  $$\Phi_m : \mathcal{X} \rightarrow \mathbb{R}^m \quad x \mapsto (k(x_1, x), \ldots, k(x_m, x))^T$$

Empirical Kernel Map

- $\Phi_m(x_1), \ldots, \Phi_m(x_m)$ contains all necessary information about $\Phi(x_1), \ldots, \Phi(x_m)$
- Gram matrix $G_{ij} = \langle \Phi_m(x_i), \Phi_m(x_j) \rangle$ satisfies $G = K^2$, where $K_{ij} = k(x_i, x_j)$

- Whitened (Kernel PCA) map:
  $$\Phi_w : \mathcal{X} \rightarrow \mathbb{R}^m \quad x \mapsto K^{-\frac{1}{2}}(k(x_1, x), \ldots, k(x_m, x))^T$$
  - satisfies $\langle \Phi_w'(x), \Phi_w'(x') \rangle = k(x, x')$

Kernel PCA map

- Data-dependent feature map into m-dimensional space
- If $K$ is invertible, $\Phi_w$ computes coordinates of $\Phi(x)$ in subspace spanned by $\Phi(x_1), \ldots, \Phi(x_m)$
- Use this coordinates as new features for any regression or classification algorithm
  - Kernelized version of PCA (principal components in feature space)
  - Similar optimality criteria as linear PCA (can be used for denoising etc.)
Data Dependent Kernel Map

- Suppose we have distinct training points \( x_1, \ldots, x_m \) and a kernel \( k \), such that the kernel matrix is p.d.
- \( K \) can be diagonalized: \( K = S D S^T \) with an orthogonal matrix \( S \) and nonnegative diagonal matrix \( D \).
- \( k(x, x') = k(x, (DS^T)x') = \langle k(x, DS^T), DS^T \rangle \).
- \( S_i \) is the \( i \)-th row of \( S \).
- Thus the map \( \Phi(x) = \sqrt{DS} \) allows the definition of a dot product in \( m \)-dimensional space for the training points \( x_1, \ldots, x_m \).

Implication

- Given training data \( x_1, \ldots, x_m \) and a kernel \( k \) which gives rise to a p.d. kernel matrix \( K \),
- We can always construct a feature space of dimension at most \( m \) that we are implicitly working in when using kernels.
- Even if \( K \) is not p.d. in general, it is sufficient that \( K \) is p.d. for the training points, and so the kernel algorithm will still work.
  - In principle, any p.d. matrix \( K \) can be used as kernel matrix (without defining \( k \)).

Properties of Kernels

- Let \( k_1 \) and \( k_2 \) be kernels then so are
  - \( \alpha k_1 \) for \( \alpha > 0 \)
  - \( k_1 + k_2 \)
  - \( k(x, x') = \lim_{n \to \infty} k_n(x, x') \), provided it exists
  - \( k(A, B) = \sum_{x \in A, x' \in B} k_n(x, x') \) for finite subsets of \( X \)
  - \( P(k_1) \) where \( P \) is a polynomial
  - \( \exp(k_1) \)
  - ...
Examples of Kernels

- Polynomial Kernel:
  - Inhomogeneous Polynomial: $k(x, x') = (\langle x, x' \rangle + c)^d$
- Gaussian Kernel:
- Sigmoid Kernel:

Prior knowledge of the problem helps in designing the right kernel for sophisticated problems (e.g., bioinformatics, text categorization, etc.)

How is similarity between inputs defined?

RBF Kernels

- $k(x, x') = f(d(x, x'))$
  - Kernel depends only on distance
  - Translation invariant
  - Unitary invariant: $k(x, x') = k(Ux, Ux')$ if $U$ is unitary
- Gaussian kernel is a special RBF kernel
  - $k(x, x) = 1$
  - All points lie in the same orthant in feature space
  - Angle between any two mapped points is $\leq \pi/2$
  - Kernel matrix has full rank, i.e., $\Phi(x_i)$ are lin. independent
  - Feature space has infinite dimension

Example: Kernel for Sequences

- Input space: all finite substrings from alphabet $A$
- Feature Map: $\Phi_i(s) = 1$ if substring $i$ is present in sequence $s$
- Dot product: count common substrings
  - Exponentially many coordinates
  - By using recursions, it can be computed in linear time
Sequence-Kernel Recursion

- \( k(s, \varepsilon) = 1 \), where \( \varepsilon \) is the empty string
- \( k(sa, t) = k(s, t) + \sum_{i} k(s, t[i:i-1])[t_i = a] \)
  - \( s, t \) ... generic sequences
  - \( a \) ... generic symbol
  - Symmetry gives \( k(s, ta) = \ldots \)

- Dynamic programming techniques evaluate this in linear time

Example

\[
 k(sa, t) = k(s, t) + \sum_{i} k(s, t[i:i-1])[t_i = a]
\]

- \( s = \text{ABBCCBBA}, t = \text{BBABBCAB} \)

- \( k(\varepsilon, A) = 1 \)
- \( k(A, A) = k(\varepsilon, A) = k(\varepsilon, \varepsilon) = 2 \)
- \( k(\varepsilon, B) = k(\varepsilon, B) = 0 \)
- \( k(A, B) = k(\varepsilon, A) + k(\varepsilon, B) = 1 \)
- \( k(\varepsilon, B) = k(\varepsilon, B) = 0 \)
- \( k(A, AB) = k(A, B) = 2 \)
- \( k(A, BB) = k(A, B) + k(A, \varepsilon) + k(\varepsilon, B) = 2 \)
- \( k(AB, BB) = k(A, AB) + k(A, B) + k(\varepsilon, \varepsilon) = 3 \)
- And so on...

Summary

- **Kernel**: similarity measure of inputs
- **Kernel Trick**: Compute high-dimensional dot products without computing feature map
  - Any algorithm that only builds upon dot products can be kernelized
- **Large margin** guarantees good generalization ability of high-dimensional hyperplanes
- **Positive definite** kernels correspond to inner products in some high-dimensional feature space
- Any positive definite kernel does the job, but performance depends heavily on choice of kernel
References

- Bernhard Schölkopf: Introduction to Kernel Methods, Analysis of Patterns Workshop, Erice, Italy, 2005
- Schölkopf & Smola: Learning with Kernels, MIT Press, 2002
- Nello Cristianini: Kernel Methods for Pattern Analysis, Talk at Erice, Italy