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
  - \( k : X \times X \rightarrow \mathbb{R} \)
  - \((x, x') \rightarrow k(x, x')\)

- for example, if \( X = \mathbb{R}^N \): canonical dot product
  \[
  k(x, x') = \sum_{i=1}^{N} x_i \cdot x'_i
  \]

- if \( X \) is not a dot product space: assume that \( k \) has a representation as a dot product in a linear space \( H \)
  - i.e., there exists a map \( \Phi : X \rightarrow H \) such that
    \[
    k(x, x') = \langle \Phi(x), \Phi(x') \rangle
    \]

- 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.

$$c_+ := \frac{1}{m_+} \sum_{y_i=1} \Phi(x_i), \quad c_- := \frac{1}{m_-} \sum_{y_i=-1} \Phi(x_i)$$

Compute sign of dot product between $w := c_+ - c_-$ and $x - c$

- Corresponds to angle
Example of a Kernel Algorithm

- $f(x) = \text{sgn}\langle (\Phi(x) - c), w \rangle = \text{sgn}\langle (\Phi(x) - (c_+ + c_-) / 2), (c_+ - c_-) \rangle = \text{sgn}\langle (\Phi(x), c_+) - \langle \Phi(x), c_- \rangle + b \rangle$
  
  $b = \frac{1}{2} \left( \|c_-\|^2 - \|c_+\|^2 \right)$

- $f(x) = \text{sgn}\left( \frac{1}{m_+} \sum_{\{i \mid y_i = +1\}} \langle \Phi(x), \Phi(x_i) \rangle - \frac{1}{m_-} \sum_{\{i \mid y_i = -1\}} \langle \Phi(x), \Phi(x_i) \rangle + b \right) = \text{sgn}\left( \frac{1}{m_+} \sum_{\{i \mid y_i = +1\}} k(\Phi(x), \Phi(x_i)) - \frac{1}{m_-} \sum_{\{i \mid y_i = -1\}} k(\Phi(x), \Phi(x_i)) + b \right)$
  
  $b = \frac{1}{2} \left( \frac{1}{m_-^2} \sum_{\{i, j \mid y_i = y_j = -1\}} k(x_i, x_j) - \frac{1}{m_+^2} \sum_{\{i, j \mid y_i = y_j = +1\}} k(x_i, x_j) \right)$

- Decision function is hyperplane in feature space
Relation to Parzen Windows

- Assume $b=0$ and $\forall x'$
  \[ \int_X k(x, x') \, dx = 1 \]
  - 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
  \[ p_{\pm}(x) = \frac{1}{m_{\pm}} \sum_{\{y_i = \pm 1\}} k(x, x_i) \]
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

\[ \Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3 \]

\[ (x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2} x_1 x_2, x_2^2) \]

- 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:
  \[ \langle \mathbf{w}, \mathbf{x} \rangle + b = 0 \]

- Decision function:
  \[ f(\mathbf{x}) = \text{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle + b) \]

- One of the simplest possible classifiers
Optimal Hyperplanes

- Maximize margin of separation
- Creates robust classifier
Motivation from Statistical Learning Theory

- Set of decision functions $f(x): X \rightarrow \{\pm 1\}$
- 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
  $$R[f] = \frac{1}{2} \int_X c(x, y, f(x)) dP(x, y)$$
- Training error = Empirical Risk
  $$R_{emp}[f] = \frac{1}{2m} \sum_{i=1}^{m} c(x_i, y_i, f(x_i))$$
- 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 $R_{\text{emp}}$ for VC-dim $h < m$, independent of distribution $P$:
  - With prob. $1-\delta$ for drawing $m$ training points:
    - $R[f] \leq R_{\text{emp}}[f] + \sqrt{\frac{1}{m} \left( h \left( \ln \frac{2m}{h} + 1 \right) + \ln \frac{4}{\delta} \right)}$
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.

![Graph showing test error and VC-dim on USPS Set]
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?
Kernel Trick 1

\[ \Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3 \]

\[(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2} x_1 x_2, x_2^2) \]

\[ \langle \Phi(x), \Phi(x') \rangle = (x_1^2, \sqrt{2} x_1 x_2, x_2^2)(x_1'^2, \sqrt{2} x_1' x_2', x_2'^2)^\top \]

\[ = \langle x, x' \rangle^2 \]

\[ = : k(x, x') \]
Kernel Trick 2

More generally: \( x, x' \in \mathbb{R}^N, d \in \mathbb{N} \):

\[
\langle x, x' \rangle^d = \left( \sum_{j=1}^{N} x_j \cdot x'_j \right)^d
= \sum_{j_1, \ldots, j_d=1}^{N} x_{j_1} \cdots x_{j_d} \cdot x'_{j_1} \cdots x'_{j_d} = \langle \Phi(x), \Phi(x') \rangle,
\]

where \( \Phi \) maps into the space spanned by all ordered products of \( d \) input directions.

- 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.
From Kernels to Feature Spaces

- 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

\[
\sum_{i,j} a_i a_j K_{ij} \geq 0 \quad K_{ij} = k(x_i, x_j)
\]

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 \textit{kernel} comes from integral operators
  \[ (T_k f)(x) = \int_X k(x, x') f(x') \, dx' \]
Reproducing Kernel Map

- For p.d. kernel $k$ and nonempty set $X$
- Define feature map $\Phi: X \rightarrow \mathbb{R}^X = \{f: X \rightarrow \mathbb{R}\}$

$\Phi(x)(.) = k(., x)$

- 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 $\langle ., . \rangle: H \times H \to \mathbb{R}$
    
    \[
    \begin{align*}
    -\langle x, x' \rangle &= \langle x', x \rangle \\
    -\langle (\lambda x + \lambda' x'), x'' \rangle &= \lambda \langle x, x'' \rangle + \lambda' \langle x', x'' \rangle \\
    -\langle x, x \rangle &\geq 0 \text{ and } \langle x, x \rangle = 0 \text{ only for } x = 0
    \end{align*}
    \]

- **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: $\forall \varepsilon > 0 \exists n \in N : \forall n', n'' > n : \|x_{n'} - x_{n''}\| < \varepsilon$
    - 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

\[ f(.) = \sum_{i=1}^{m} \alpha_i k(.,x_i) \]

\[ g(.) = \sum_{j=1}^{m'} \beta_j k(.,x'_j) \]

\( m, m' \in N, \alpha_i, \beta_j \in R, x_1,...,x_m, x'_1,...,x'_m' \in X \)

- These elements form a vector space
Define Dot Product

\[ \langle f, g \rangle := \sum_{i=1}^{m} \sum_{j=1}^{m'} \alpha_i \beta_j k(x_i, x_j') \]

\[ = \sum_{i=1}^{m} \alpha_i g(x_i) = \sum_{j=1}^{m'} \beta_j f(x_j') \]

- well-defined, although coefficients need not be unique
- Symmetric
- Bilinear
- Is it positive definite, i.e. \( \langle f,f \rangle \geq 0 \)?
Dot Product

\[ \langle f, f \rangle = \sum_{i,j=1}^{m} \alpha_i \alpha_j k(x_i, x_j) \geq 0 \quad \text{because } k \text{ is p.d.} \]

\[ \sum_{i,j} \gamma_i \gamma_j \langle f_i, f_j \rangle = \left\langle \sum_i \gamma_i f_i, \sum_j \gamma_j f_j \right\rangle = \langle f, f \rangle \geq 0 \]

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

Thus \( \langle \ldots \rangle \) is itself a p.d. kernel on the function space \( H \)
Reproducing Kernels

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

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

- Last step to show that \( \langle ., . \rangle \) 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
  \[ p_x : H \rightarrow R \]
  \[ f \mapsto p_x(f) = f(x) \]
  exist and are continuous (i.e. when $f$ and $f'$ are close in $H$, $f(x)$ and $f'(x)$ are close in $R$).

- In that case for each $x' \in X$ there exists a unique function of $x$, called $k(x,x')$ s.t.
  \[ f(x') = \langle f, k(.,x') \rangle \]
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 \( L_2(\mathcal{X}) \) (where \( \mathcal{X} \) is some compact space),

\[
\int_{\mathcal{X}} k(x, x') f(x) f(x') \, dx \, dx' \geq 0,
\]

it can be expanded as

\[
k(x, x') = \sum_{i=1}^{\infty} \lambda_i \psi_i(x) \psi_i(x')
\]

using eigenfunctions \( \psi_i \) and eigenvalues \( \lambda_i \geq 0 \).
Mercer Feature Map

In that case

\[ \Phi(x) := \left( \sqrt{\lambda_1} \psi_1(x) \right) \begin{pmatrix} \sqrt{\lambda_2} \psi_2(x) \\ \vdots \end{pmatrix} \]

satisfies \( \langle \Phi(x), \Phi(x') \rangle = k(x, x') \).

Proof:

\[
\langle \Phi(x), \Phi(x') \rangle = \left\langle \left( \sqrt{\lambda_1} \psi_1(x) \right) \begin{pmatrix} \sqrt{\lambda_2} \psi_2(x) \\ \vdots \end{pmatrix}, \left( \sqrt{\lambda_1} \psi_1(x') \right) \begin{pmatrix} \sqrt{\lambda_2} \psi_2(x') \\ \vdots \end{pmatrix} \right\rangle
\]

\[
= \sum_{i=1}^{\infty} \lambda_i \psi_i(x) \psi_i(x') = k(x, x')
\]
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

\[
f(x) = \sum_{i=1}^{\infty} \alpha_i k(x, x_i) = \sum_{i=1}^{\infty} \alpha_i \sum_{j=1}^{\infty} \lambda_j \psi_j(x) \psi_j(x_i)
\]

\[
\langle f, k(., x') \rangle = \sum_{i=1}^{\infty} \alpha_i \sum_{j, n=1}^{\infty} \lambda_j \psi_j(x_i) \langle \psi_j, \psi_n \rangle \lambda_n \psi_j(x') =
\]

\[
= \sum_{i=1}^{\infty} \alpha_i \sum_{j=1}^{\infty} \lambda_j \psi_j(x') \psi_j(x_i) = f(x')
\]

by letting $\langle \psi_j, \psi_n \rangle = \delta_{jn} / \lambda_j$

\[
\delta_{jn} = \begin{cases} 
1 & j = n \\
0 & \text{else}
\end{cases}
\]
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 \( \Phi_1(x) \neq \Phi_2(x) \)
    - 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:**

  $\Phi_m : \mathcal{X} \rightarrow \mathbb{R}^m$
  $x \mapsto [k(., x) | (x_1, \ldots, x_m) = (k(x_1, x), \ldots, k(x_m, x))^\top$
Empirical Kernel Map

\[ \Phi_m : \mathcal{X} \rightarrow \mathbb{R}^m \]
\[ x \mapsto k(., x)|_{(x_1, \ldots, x_m)} = (k(x_1, x), \ldots, k(x_m, x))^\top \]

- \( \Phi_m(x_1), \ldots, \Phi_m(x_m) \) 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:

\[ \Phi^w_m : \mathcal{X} \rightarrow \mathbb{R}^m \]
\[ x \mapsto K^{-1/2}(k(x_1, x), \ldots, k(x_m, x))^\top \]

- satisfies \( \langle \Phi^w_m(x), \Phi^w_m(x') \rangle = k(x, x') \)
Kernel PCA map

- Data-dependent feature map into m-dimensional space
- If $K$ is invertible, $\Phi_m^w$ 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.)
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 $D$.

$$k(x_i, x_j) = K_{ij} = (SDS^T)_{ij} = \langle S_i, DS_j \rangle = \langle \sqrt{DS}_i, \sqrt{DS}_j \rangle$$

- $S_i$ is the $i$-th row of $S$

Thus the map $\Phi(x_i) = \sqrt{DS}_i$ allows the definition of a dot product in $m$-dimensional space for the training points $x_1, \ldots, x_m$. 

Data Dependent Kernel Map
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 \( \alpha > 0 \)
  - \( k_1 + k_2, \, k_1 \cdot k_2 \)
  - \( k(x, x') := \lim_{n \to \infty} k_n(x, x') \), provided it exists
  - \( k(A, B) := \sum_{x \in A, x' \in B} k_1(x, x') \) \( A, B \): finite subsets of \( X \)
  - \( P(k_1) \) where \( P \) is a polynomial
  - \( \exp(k_1) \)
  - \( \ldots \)
Examples of Kernels

- Polynomial Kernel:
  \[ k(x, x') = \langle x, x' \rangle^d \]
- Inhomogeneous Polynomial:
  \[ k(x, x') = \left( \langle x, x' \rangle + c \right)^d \]
- Gaussian Kernel:
  \[ k(x, x') = \exp \left( -\frac{||x - x'||^2}{2\sigma^2} \right) \]
- Sigmoid Kernel:
  \[ k(x, x') = \tanh(\kappa \langle x, x' \rangle + \theta) \]

- 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 \(< \pi/2\)
  - 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
- \( k(sa, t) = k(s, t) + \sum_i k(s, t[1:i-1])[t_i = a] \)
  - \( s, t \) … generic sequences
  - \( a \) … generic symbol
  - Symmetry gives \( k(s, ta) = \ldots \)

- Dynamic programming techniques evaluate this in linear time
Example

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

- \( s = ABBCBBCA, \ t = BBABBCAB \)
  - \( k(\varepsilon, t) = k(s, \varepsilon) = 1 \)
  - \( K(A, A) = k(\varepsilon, A) + k(\varepsilon, \varepsilon)[t_1 = A] = 2 \)
  - \( k(A, B) = k(\varepsilon, B) + 0 = 1 \)
  - \( k(B, B) = k(\varepsilon, B) + k(\varepsilon, \varepsilon)[t_1 = B] = 2 \)
  - \( k(A, BB) = k(\varepsilon, BB) + 0 = 1 \)
  - \( k(AB, A) = k(A, A) = 2 \)
  - \( k(AB, B) = k(A, B) + k(A, \varepsilon)[t_1 = B] = 2 \)
  - \( k(AB, BB) = k(A, BB) + k(A, \varepsilon)[t_1 = B] + k(A, B)[t_2 = B] = 3 \)
  - 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