Kernel functions for structured data

Lecture by Klaus-Robert Müller, TUB 2015

Outline

Brief Review: Kernels

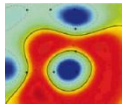
- Definition and Properties

Kernels for Strings

- Generic String Kernel
- Bag-of-words, N-grams and Substrings
- Efficient Implementation

Kernels for Trees

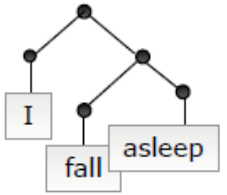
- Parse Tree Kernel
- Efficient Implementation

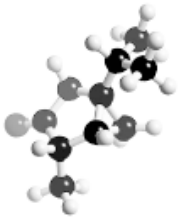


Structured Data

Structured data ubiquituous in applied sciences

- *Bioinfomatics*
  e.g. DNA and protein sequences
- *Natural language processing*
  e.g. text documents and parse trees
- *Computer security*
  e.g. network traffic and program behavior
- *Chemoinformatics*
  e.g. molecule structures and relations





Structured data $\neq$ vectors $\Rightarrow$ No machine learning?

Brief review: Kernels

What is a Kernel

Kernel function or short kernel:

- A positive semi-definite function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$
- Similarity measure for objects in a domain $\mathcal{X}$
- Basic building block of many learning algorithms

Definition

A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a *kernel* iff k is symmetric and positive semi-definite for any subset $\{x_1, \ldots, x_l\} \subset \mathcal{X}$, that is

$$\sum_{i,j=1}^{m} c_i c_j k(x_i, x_j) \geq 0 \quad \text{with} \quad c_1, \ldots, c_m \in \mathbb{R}.$$

Kernels and Feature Spaces

Theorem

A kernel k induces a feature map $\psi : \mathcal{X} \to \mathcal{F}$ to a Hilbert space, where k equals an inner product. That is, for all $x, y \in \mathcal{X}$

$$k(x, y) = \langle \psi(x), \psi(y) \rangle .$$

Interface to geometry in feature space

- Access to inner products, vector norms and distances, e.g.,

$$||\psi(x)||_2 = \sqrt{k(x, x)}$$
$$||\psi(x) - \psi(y)||_2 = \sqrt{k(x, x) + k(y, y) - 2k(x, y)}$$

Classic Kernels

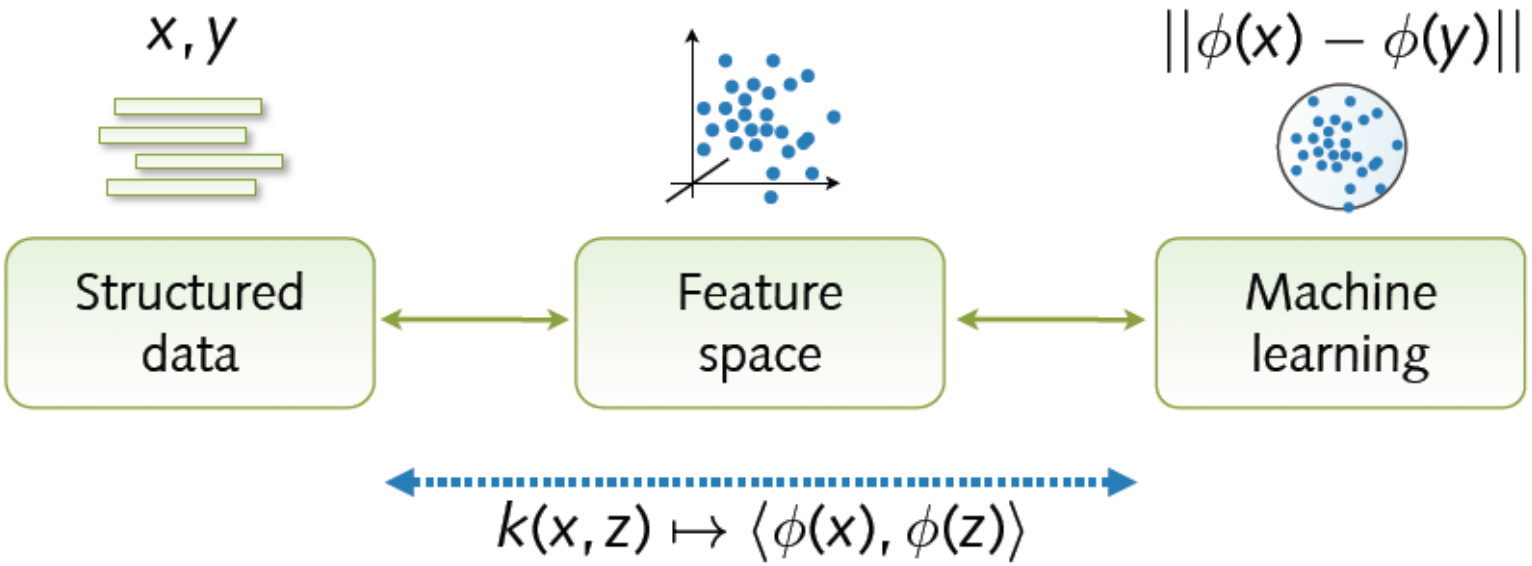
Let $\mathcal{X} \subseteq \mathbb{R}^d$. Then kernels $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ are given by

- $k(x, y) := \langle x, y \rangle = \sum_{i=1}^{d} x_i y_i$ (Linear kernel)
- $k(x, y) := (\langle x, y \rangle + \theta)^p$ (Polynomial kernel)
- $k(x, y) := \exp\left(-\frac{||x-y||^2}{\gamma}\right)$ (Gaussian kernel)
- $k(x, y) := \tanh(\langle x, y \rangle + \theta)$ (Sigmoidal kernel)

However: Domain $\mathcal{X}$ not restricted to vectorial data!

Kernels for structured data

- Definition of kernel k over non-vectorial domain $\mathcal{X}$
- Any k valid, if symmetric and positive semi-definite
- Integration with kernel-based learning methods



Kernels for Strings

Strings

Alphabet

An alphabet $\mathcal{A}$ is a finite set of discrete symbols

- DNA, $\mathcal{A} = \{A,C,G,T\}$
- Natural language text, $\mathcal{A} = \{a,b,c,\ldots A,B,C,\ldots\}$

String or Sequence

A string x is concatenation of symbols from $\mathcal{A}$

- $\mathcal{A}^n$ = all strings of length n
- $\mathcal{A}^*$ = all strings of arbitary length
- $|x|$ = length of a string

Embedding Strings

Mapping of strings to a feature space

- Characterize strings using a *language* $L \subseteq \mathcal{A}^*$.
- Feature space spanned by occurences of words $w \in L$

Feature map

A function $\phi : \mathcal{A}^* \to \mathbb{R}^{|L|}$ mapping strings to $\mathbb{R}^{|L|}$ given by

$$\phi : x \longmapsto \left(\#_w(x) \cdot \sqrt{N_w} \right)_{w \in L}$$

where $\#_w(x)$ returns the occurences of w in string x and N_w is a weighting of individual words.

String Kernels

Generic String Kernel

A generic string kernel $k : \mathcal{A}^* \times \mathcal{A}^* \to \mathbb{R}$ is given by

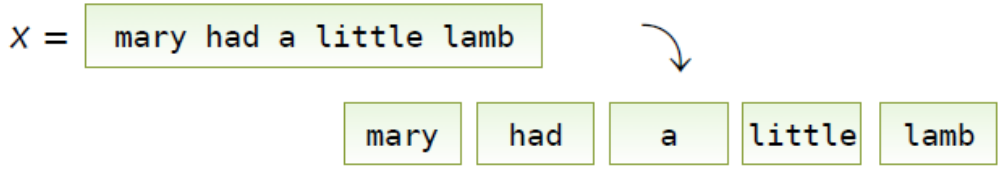
$$k(x, z) = \langle \phi(x), \phi(z) \rangle = \sum_{w \in L} \#_w(x) \cdot \#_w(z) \cdot N_w$$

Proof.

By definition k is an inner product in $\mathbb{R}^{|L|}$ and thus symmetric and positive semi-definite. □

Bag-of-Words

Characterization of strings using non-overlapping substrings



Bag-of-Words Kernel

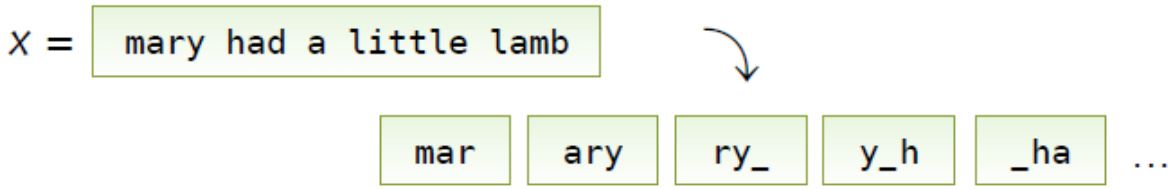
String kernel using embedding language of *words* with delimiters D

$$k(x,y) = \sum_{w \in L} \#_w(x) \cdot \#_w(y) \cdot N_w \quad \text{with} \quad L = (\mathcal{A} \setminus D)^*$$

- Suitable for analysis of strings with known structure
  e.g., natural language text, tokenized data, log files

N-grams

Characterization of strings using substrings of length n



N-gram Kernel

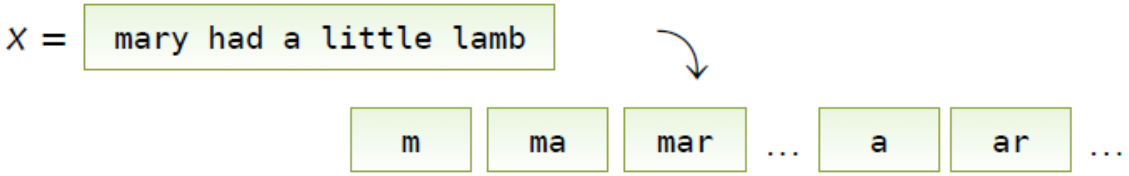
String kernel using embedding language of *n-grams*:

$$k(x,y) = \sum_{w \in L} \#_w(x) \cdot \#_w(y) \cdot N_w \quad \text{with} \quad L = \mathcal{A}^n$$

- Suitable for analysis of strings with unknown structure, e.g., DNA sequences, network attacks, binary data

All Substrings

Characterization of strings using all possible substrings



All-Substring Kernel

String kernel using embedding language of *all strings*:

$$k(x,y) = \sum_{w \in L} \#_w(x) \cdot \#_w(y) \cdot N_w \quad \text{with} \quad L = \mathcal{A}^*$$

- Suitable for analysis of generic string data
- Encoding of prior knowledge in weighting N_w

Implementing String Kernels

Efficient computation of string kernel $k(x, z)$

- Feature space high-dimensional but sparsely populated
- Sufficient to consider only w with $\#_w(x) \neq 0$ and $\#_w(z) \neq 0$
- Application of special data structures for strings

Implementation strategies

1. Explicit but sparse representation of feature vectors,
   $\longrightarrow$ hash tables, tries and sorted arrays
2. Implicit representation of feature vectors,
   $\longrightarrow$ suffix trees and arrays

Sorted Arrays

Example: strings x and y with embedding language $L = \mathcal{A}^3$

$x =$ abbaa $\quad z =$ baaaab

Extracted words w stored with $\#_w(x)$ in sorted array

$\phi(x)$ — aa|1 — ab|1 — ba|1 — bb|1

$\phi(z)$ — aa|3 — ab|1 — ba|1

- Explicit kernel computation $\longrightarrow$ parallel loop over arrays
- Run-time $\mathcal{O}(|x| + |z|)$ for words with no or bounded overlap

Tries

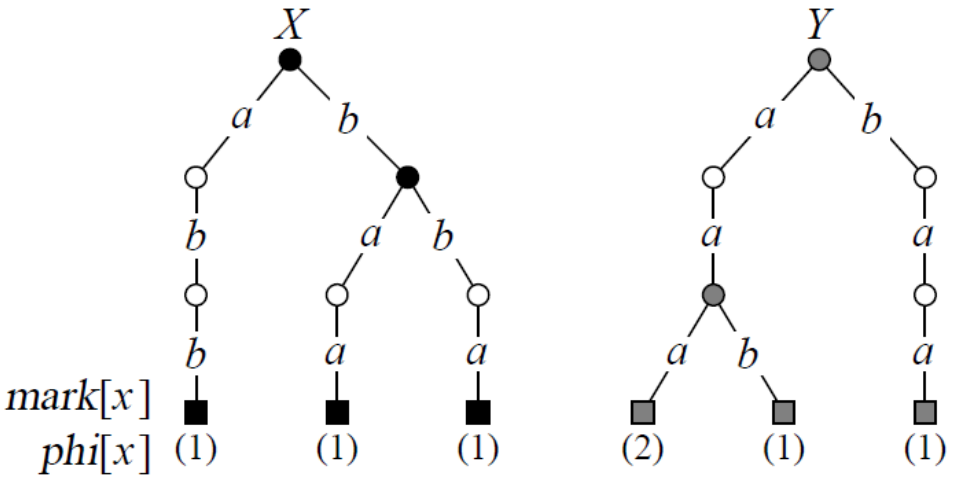
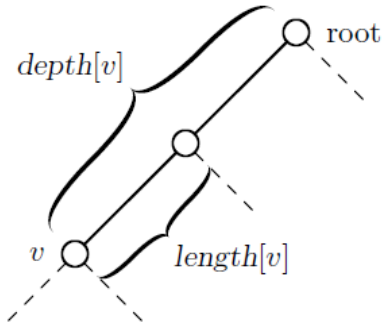


Figure 2: Tries of 3-grams for $\mathbf{x} = abbaa$ and $\mathbf{y} = baaaab$. The number in brackets at leaves indicate the number of occurrences. Marked nodes are squared. White nodes are implicit and not maintained in a compact trie representation.

Algorithm: to compare, recursively traverse both tries in parallel $\mathcal{O}(|x| + |z|)$

Suffix Trees

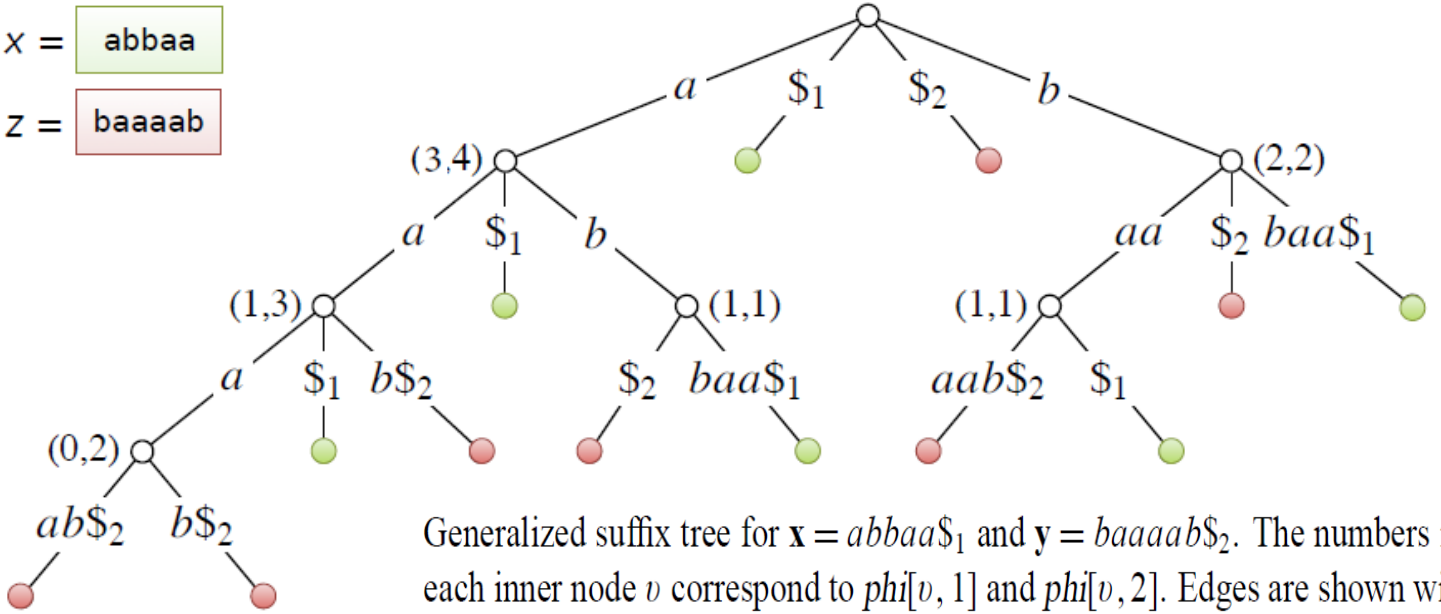
Data structure. A *generalized suffix tree* (GST) is a compact trie containing all suffixes of a set of sequences $\mathbf{x}_1, \ldots, \mathbf{x}_l$ (Gusfield, 1997). Every path in a GST from the root to a leaf corresponds to one suffix. A GST is obtained by extending each sequence $\mathbf{x}_i$ with a delimiter $\$_i \notin \mathcal{A}$ and constructing a suffix tree from the concatenation $\mathbf{z} = \mathbf{x}_1\$_1 \ldots \mathbf{x}_l\$_l$.



For each GST node v we denote by *children*$[v]$ the set of child nodes, by *length*$[v]$ the number of symbols on the incoming edge, by *depth*$[v]$ the total number of symbols on the path from the root node to v and by *phi*$[v, i]$ the number of suffixes of $\mathbf{x}_i$ passing through node v. As every subsequence of $\mathbf{x}_i$ is a prefix of some suffix, *phi*$[v, i]$ reflects the occurrences (alternatively frequency or binary flag) for all subsequences terminating on the edge to v. An example of a GST is given in Figure 3. In the remaining part we focus on the case of two sequences $\mathbf{x}$ and $\mathbf{y}$ delimited by $\$_1$ and $\$_2$, computation of similarity measures over a set of sequences being a straightforward extension.

Suffix Trees

Strings jointly stored in generalized suffix tree



Generalized suffix tree for $\mathbf{x} = abbaa\$_1$ and $\mathbf{y} = baaaab\$_2$. The numbers in brackets at each inner node v correspond to $phi[v, 1]$ and $phi[v, 2]$. Edges are shown with associated subsequences instead of indices.

- Implicit kernel computation $\longrightarrow$ depth first traversal
- Run-time $\mathcal{O}(|x| + |z|)$ for arbitrary embedding languages

Kernels for Trees

Trees and Parse Trees

Tree

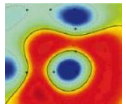
A tree $x = (V, E, v^*)$ is an acyclic graph (V, E) rooted at $v^* \in V$.

Parse tree

A tree x deriving from agrammar, such that each node $v \in V$ is associated with a production rule $p(v)$.

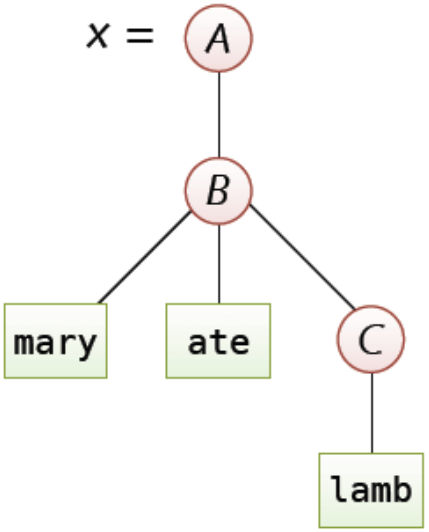
Further notation

- v_i = i-th child of node $v \in V$
- $|v|$ = number of children of $v \in V$
- T = set of all possible parse trees



Parse Trees

Tree representation of "sentences" derived from a grammar



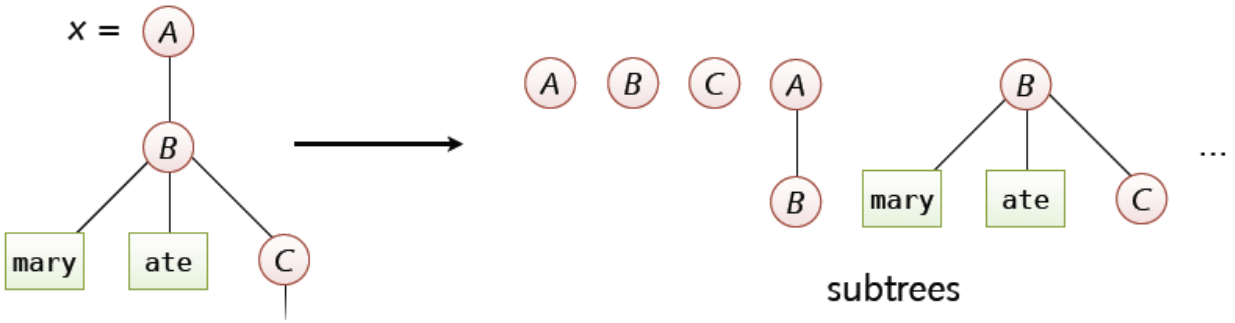
Parse tree for "mary ate lamb" with production rules

- $p_1 : A \longrightarrow B$
- $p_2 : B \longrightarrow$ "mary" "ate" C
- $p_3 : C \longrightarrow$ "lamb"

Common data structure in several application domains, e.g., natural language processing, compiler design, ...

EmbeddingTrees

Characterization of parse trees using contained subtrees



Feature map

A function $\phi : T \to \mathbb{R}^{|T|}$ mapping trees to $\mathbb{R}^{|T|}$ given by

$$\phi : x \longmapsto (\#_t(x))_{t \in T}$$

where $\#_t(x)$ returns the occurences of subtree t in x.

Parse Tree Kernel

Parse Tree Kernel

A tree kernel $k : T \times T \to \mathbb{R}$ is given by

$$k(x, z) = \langle \phi(x), \phi(z) \rangle = \sum_{t \in T} \#_t(x) \cdot \#_t(z)$$

Proof.

By definition k is an inner product in the space of all trees T and thus symmetric and positive semi-definite. □

Counting shared subtrees

Parse tree kernel and counting

- Parse tree kernel counts the number of shared subtrees
- For each pair (v, w) determine shared subtrees at v and w.

$$k(x,z) = \sum_{t \in T} \#_t(x) \cdot \#_t(z) = \sum_{v \in V_x} \sum_{w \in V_z} c(v,w)$$

Counting function

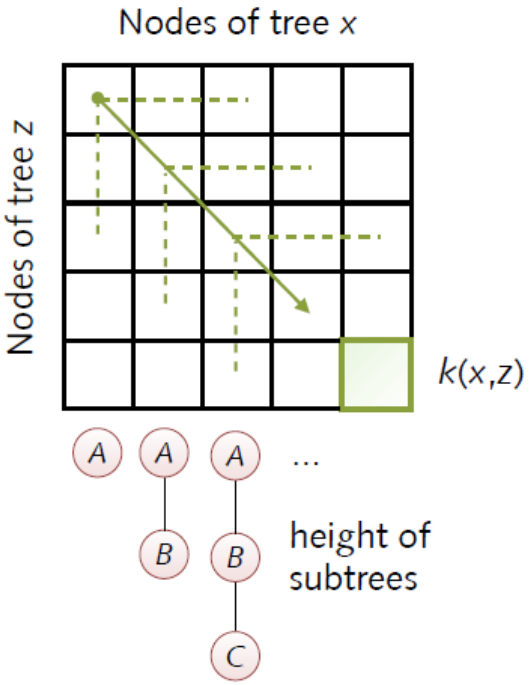
- $c(v,w) = 0$ if $p(v) \neq p(w)$ (different production)
- $c(v,w) = 1$ if $|v| = |w| = 0$ (leaf nodes)
- otherwise

$$c(v,w) = \prod_{i=1}^{|v|} (1 + c(v_i, w_i))$$

Implementation of tree kernels

Efficient implementation using dynamic programming

- Explicit feature vector representations intractable
- Implicit kernel computation by counting shared subtrees



Matrix of counts $c(v, w)$ for all shared subtrees sorted by height

- Count small subtrees first
- Gradually aggregate counts

Run-time $\mathcal{O}(|V_x| \cdot |V_z|)$.

Kernels from probabilistic models

- idea
  - make use of probabilistic modeling for discriminative training (e.g. a HMM model)
  - find natural comparison between objects (of **various** types) that is induced by a generative probability model $P(X|\Theta)$
- Thus: derive a SV-Kernel that reflects this probabilistic model!
- parametric class of models: $P(X|\theta), \theta \in \Theta$ forms a Riemannian manifold M_Θ with a metric: the Fisher Information G

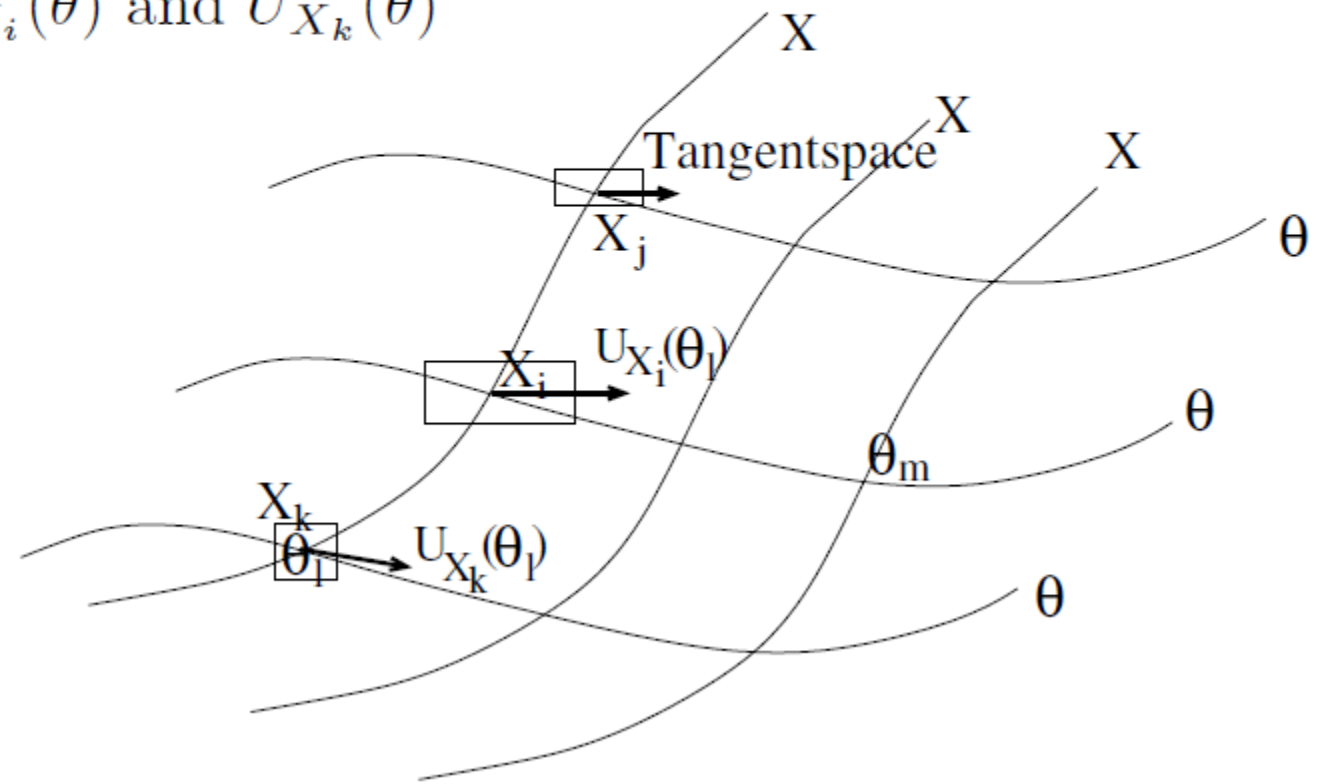
$$G = E_X\{U_X U_X^\top\}, \quad \text{where} \quad U_X = \nabla_\theta \log P(X|\theta) \quad \text{[Fisher Score]}$$

- "natural" mapping to feature space $\Phi_X = G^{-1}U_X$, so

$$K(X_i, X_j) \sim \Phi_{X_i} G \Phi_{X_j} = U_{X_i} G^{-1} U_{X_j} \quad \text{for fixed parameter } \theta$$

A geometrical picture

idea: compute similarity between X_i and X_k in terms of probabilistic model with fixed parameters θ_l between $P(X_i|\theta_l)$ and $P(X_k|\theta_l)$ by appropriately computing scalar products of tangent vectors $U_{X_i}(\theta)$ and $U_{X_k}(\theta)$



Protein superfamily classification

- 4541 sequences are hierarchically labeled into 7 classes, 558 folds, 845 superfamilies and 1343 families according to the SCOP scheme
- task: classify top category classes.
- number of sequences: 791, 1277, 1015, 915, 84, 76, 383
- classify as a set of 2-class problems
- train HMMs to obtain $p(\mathbf{x}, \Theta)$ for each class and
- construct Fisher Kernel: $K(X_i, X_j) \sim U_{X_i} G^{-1} U_{X_j}$
- **note**: FK can be improved by TOP kernel (cf. Tsuda et al. 2001 & 2002)

Top Kernel

- The new feature extractor is described as

$$\boldsymbol{f}_{\hat{\boldsymbol{\theta}}}(\mathbf{x}) := (v(\mathbf{x}, \hat{\boldsymbol{\theta}}), \partial_{\theta_1} v(\mathbf{x}, \hat{\boldsymbol{\theta}}), \ldots, \partial_{\theta_p} v(\mathbf{x}, \hat{\boldsymbol{\theta}}))^\top$$

  where

$$\begin{aligned} v(\mathbf{x}, \boldsymbol{\theta}) &= F^{-1}(P(y = +1|\mathbf{x}, \boldsymbol{\theta})) \\ &= \log(P(y = +1|\mathbf{x}, \boldsymbol{\theta})) - \log(P(y = -1|\mathbf{x}, \boldsymbol{\theta})), \end{aligned}$$

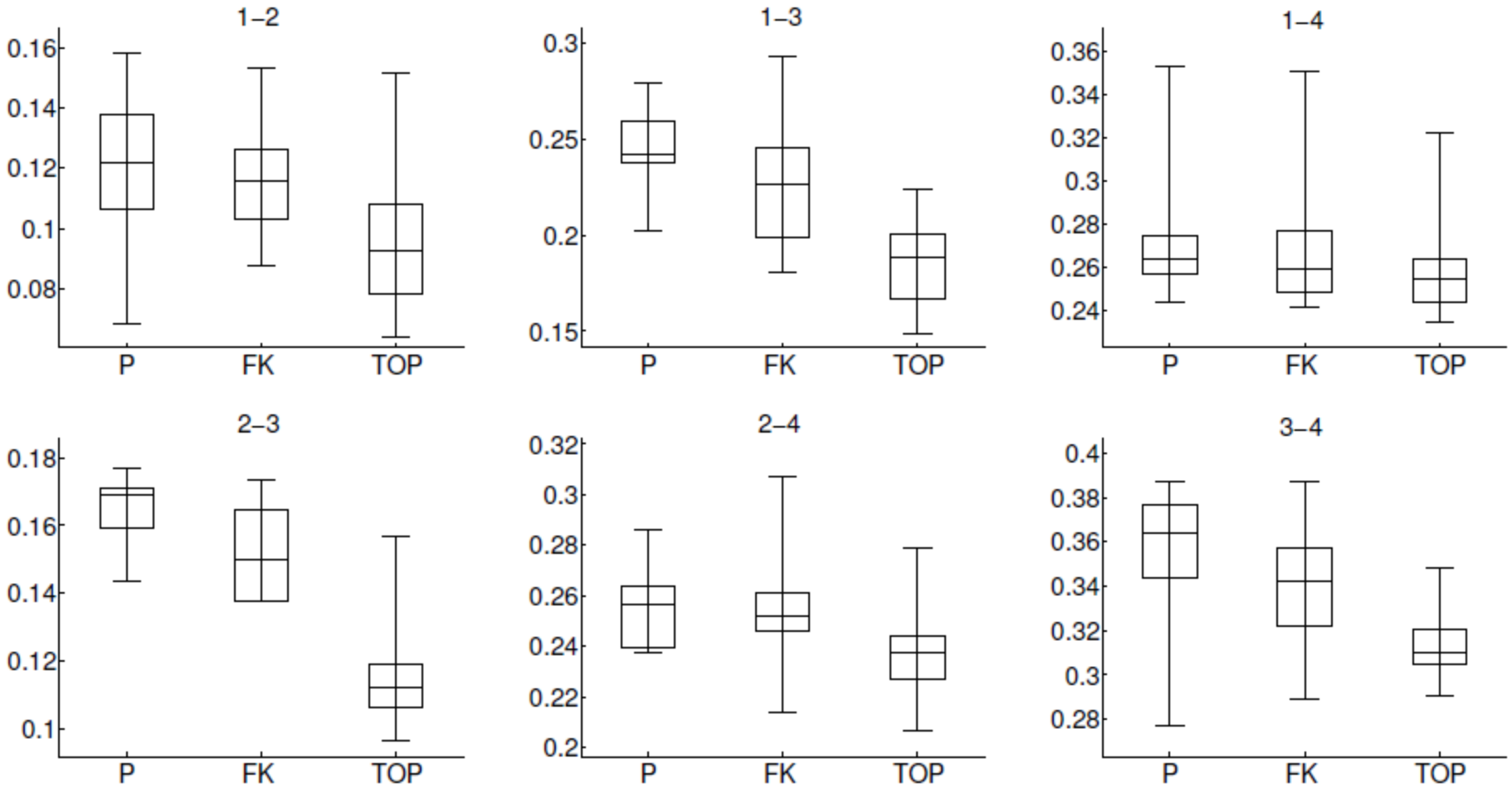
- We refer the inner product in the feature space as the *TOP kernel*

$$K_T(\mathbf{x}, \mathbf{x}') = \boldsymbol{f}_{\hat{\boldsymbol{\theta}}}(\mathbf{x})^\top \boldsymbol{f}_{\hat{\boldsymbol{\theta}}}(\mathbf{x}').$$

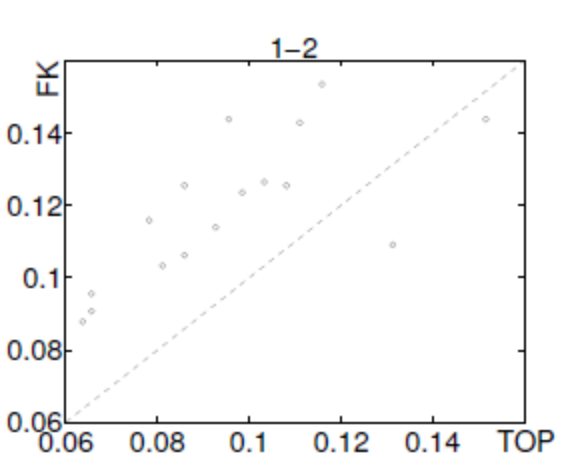
- Plug-in Estimate performs classification by means of the posterior probabilities derived from class conditional likelihoods

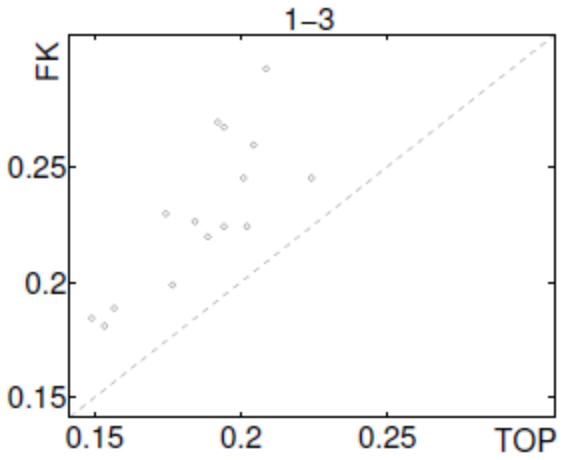
$$\hat{y} = \text{sgn}(P(y = +1|\mathbf{x}, \hat{\boldsymbol{\theta}}) - 0.5)$$

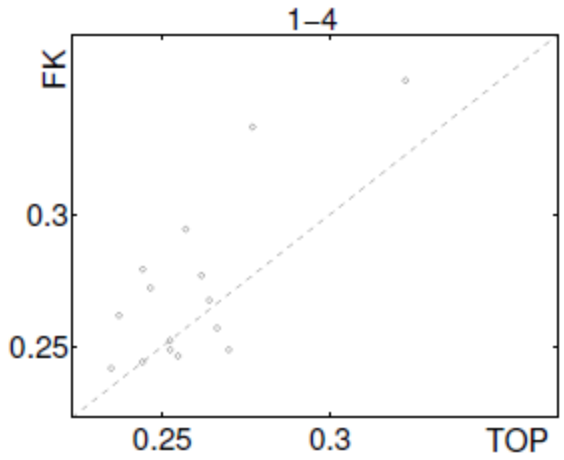
Protein superfamilies

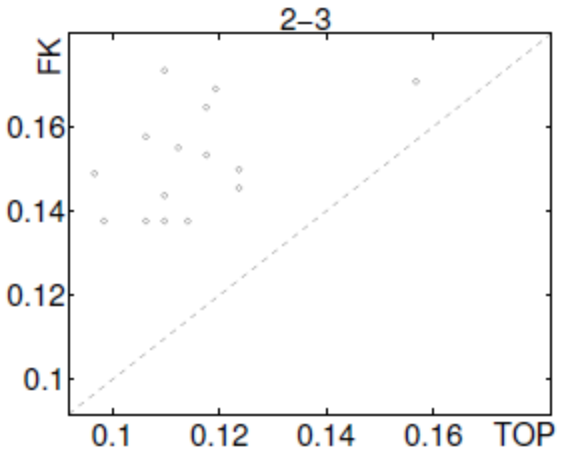


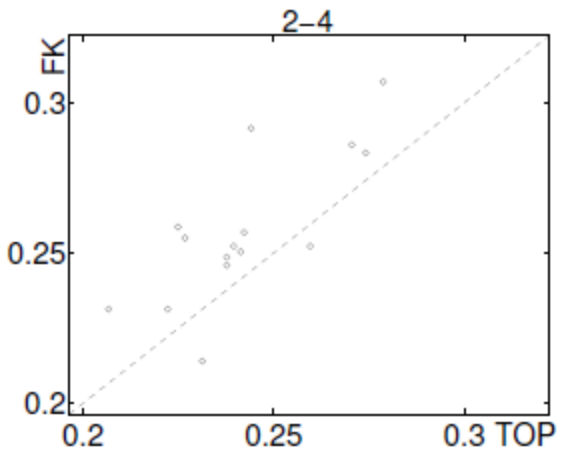
Comparison plots in protein classification

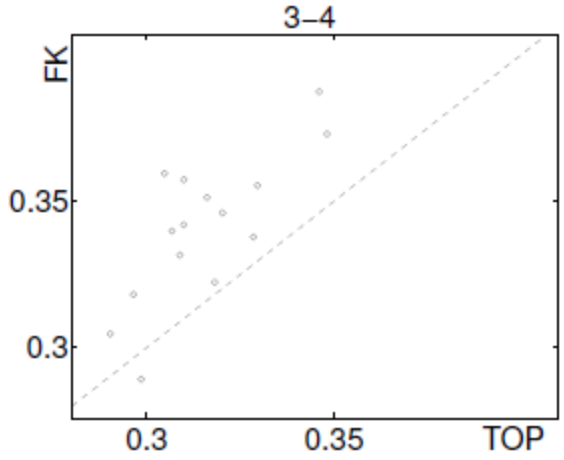




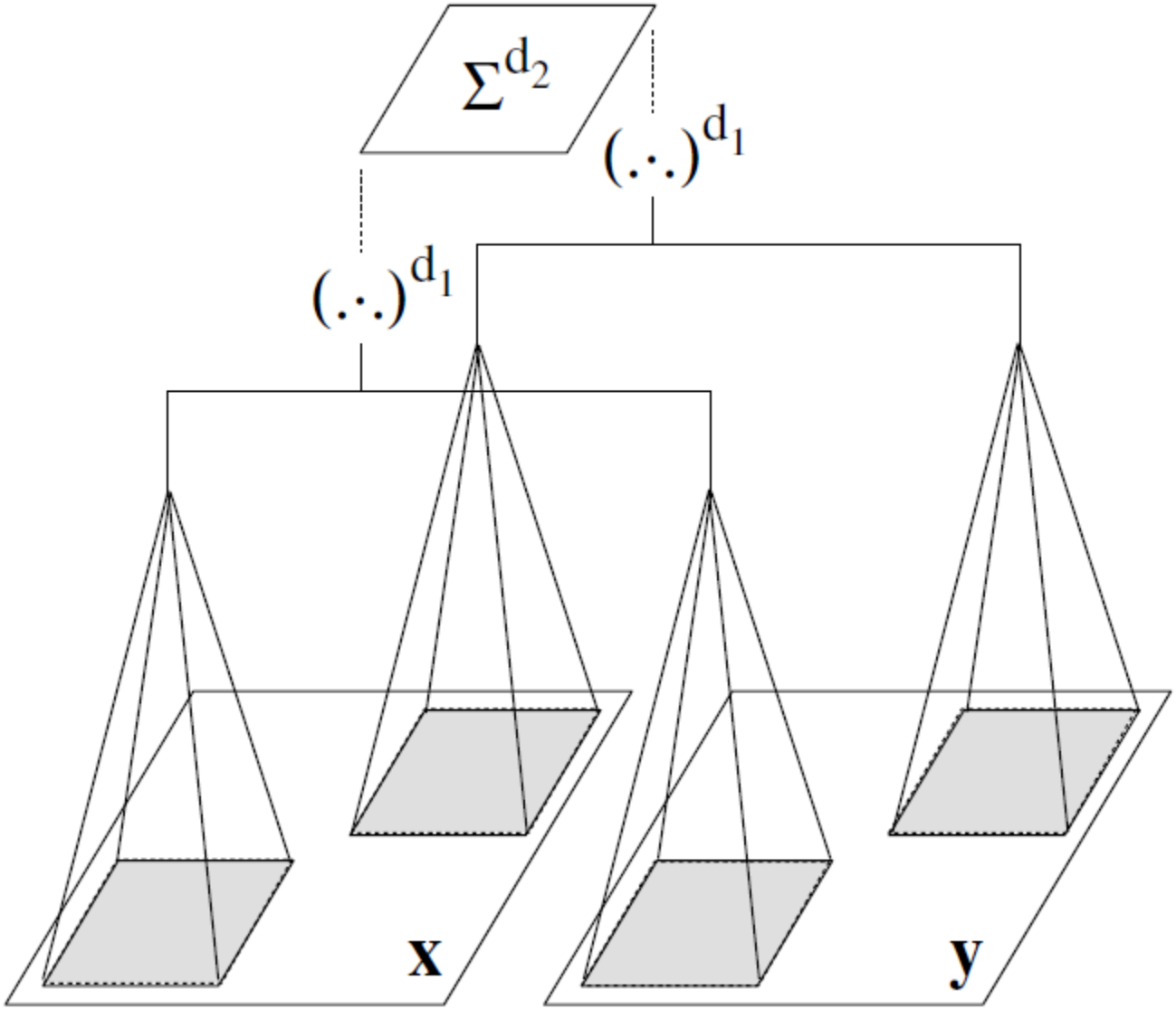








Engineering Kernels



Conclusion

Kernels for strings and trees

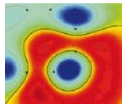
- Effective means for learning with structured data
- Several efficient kernels and implementations

More interesting kernels for structured data

- Kernel for graphs, images, sounds, ...
- Convolution kernels, approximate kernels, ...

Interesting applications (upcoming lectures)

- "Catching hackers": Network intrusion detection
- "Discovering genes": Analysis of DNA sequences



Acknowledgement

This lecture is based on a 2011 lecture at TU Berlin
prepared and given by Konrad Rieck within the ML 2 cycle.

Tsuda, K., Kawanabe, M., Rätsch, G., Sonnenburg, S., & Müller, K. R. (2002). A new discriminative kernel from probabilistic models. *Neural Computation*, *14*(10), 2397-2414.

Rieck, K., Krueger, T., Brefeld, U., and Müller, K.-R. (2010).
Approximate tree kernels.
Journal of Machine Learning Research, 11(Feb):555–580.

Rieck, K. and Laskov, P. (2008).
Linear-time computation of similarity measures for sequential data.
Journal of Machine Learning Research, 9(Jan):23–48.

Shawe-Taylor, J. and Cristianini, N. (2004).
Kernel methods for pattern analysis.
Cambridge University Press.