Kernels for Structured Data

Prof. Dr. Klaus-Robert Müller
Konrad Rieck

Vorlesung “Maschinelles Lernen – Theorie und Anwendung”
Technische Universität Berlin

April 29, 2008

Outline

1. Brief review: Kernels
   - Definition and properties
2. Kernels for Sequences
   - Sequence kernels
   - Bag-of-words and n-grams
   - Subsequences
3. Kernels for Trees
   - Parse tree kernel
   - Shallow tree kernel

Structured Data

Ubiquitous in important application domains

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

How to incorporate structure into learning methods?

What is a 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 for many learning algorithms

Definition

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

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

Classic Kernels

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

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

But: type of domain $\mathcal{X}$ not restricted to vectorial data.

Induced Feature Space

Theorem

A kernel k induces a feature map $\psi : \mathcal{X} \to \mathcal{H}$ to a Hilbert space, such that for all $x, y \in \mathcal{X}$

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

corresponds to an inner product in $\mathcal{H}$.

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

$$||\psi(x)||_2 = \sqrt{k(x, x)}$$

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

Why use Kernels for Learning?

Advantages

- Efficient computation in high-dimensional feature spaces
- Non-linear feature maps for complex decision surfaces
- Abstraction from data representation and learning methods
  $\Rightarrow$ application of learning methods to structured data

Kernel-based learning

- Classification (Support Vector Machines, Kernel Peceptron)
- Clustering (Kernel k-means, Spectral Clustering)
- Data projection (Kernel PCA, Kernel ICA)

Kernels for Sequences

Sequences

Alphabet

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

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

Sequence

A sequence x is concatenation of symbols from $\mathcal{A}$, i.e., $x \in \mathcal{A}^*$

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

Embedding Sequences

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

Feature map

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

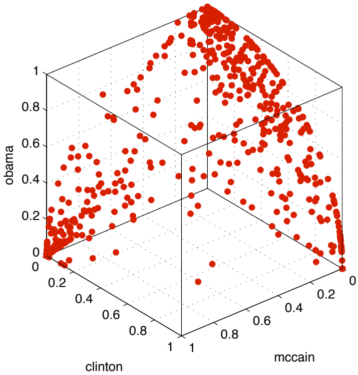
$$x \mapsto \left(\#_w(x) \cdot \sqrt{N_w}\right)_{w \in L}$$

where $\#_w(x)$ returns the frequency of w in sequence x.

- Refinement of embeddung using weighting constants N_w
- Normalization, often $||\phi(x)||_1 = 1$ or $||\phi(x)||_2 = 1$.

Example: Embedding

Embedding of new articles using the exemplary language
$L = \{\text{Barack}, \text{Clinton}, \text{Obama}\}$



Vectorial representation of sequence content via language L

Data lies on quarter-sphere due to $||\phi(x)||_2 = 1$ normalization

Source: news.google.com on 15. April 2008

Sequence Kernels

Generic Sequence Kernel

A sequence kernel $k : \mathcal{A}^* \times \mathcal{A}^* \to \mathbb{R}$ over ϕ is defined by

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

Proof.

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

- Feature space induced by ϕ *explicit* but *sparse*.
- Naive running time $\mathcal{O}(|x|^2 + |y|^2)$

Bag-of-Words

Characterization of sequences by non-overlapping words.

$$x = \text{“Hasta la vista, baby.”} \longrightarrow \{ \text{“Hasta”, “la”, “vista”, “baby”} \}$$

Bag-of-Words Kernel

Sequence kernel using embedding language containing words

$$L = \text{Dictionary (explicit)} \quad \text{or} \quad L = (\mathcal{A} \backslash D)^* \text{ (implicit)}$$

with $D \subset \mathcal{A}$ delimiter symbols, e.g., punctation and space.

- Extension using stemming techniques, “helping” ⇒ “help”
- Weighting to control contribution of words

Implementing Bag-of-Words

- Efficient realization using sorted arrays or hash tables

$$x = \text{“to be or not to be”}$$
$$\phi(x) = [\text{“be”} : 2] \rightarrow [\text{“not”} : 1] \rightarrow [\text{“or”} : 1] \rightarrow [\text{“to”} : 2]$$

- Kernel computation similar to merging lists

$$\phi(x) = [\text{“be”} : 2] \rightarrow [\text{“not”} : 1] \rightarrow [\text{“or”} : 1] \rightarrow [\text{“to”} : 2]$$
$$\phi(y) = [\text{“be”} : 1] \rightarrow [\text{“free”} : 1] \rightarrow [\text{“to”} : 1]$$
$$\longrightarrow \quad 2 \cdot 1 \quad + \quad 2 \cdot 1$$

- Run-time $\mathcal{O}(n|x| + n|y|)$ for words of length n.

N-grams

Characterization of sequences by subsequences of length n

$$x = \text{“Hasta la vista, baby.”} \longrightarrow \{ \text{“Has”}, \text{“ast”}, \text{“sta”}, \ldots \}$$

Spectrum Kernel

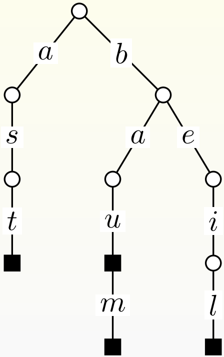
Sequence kernel using embedding language containing all sequences of length n (n-grams):

$$L = \mathcal{A}^n \text{ (normal)} \quad \text{or} \quad L = \bigcup_{i=1}^{n} \mathcal{A}^i \text{ (blended)}$$

- No prior knowledge of application domain required
- Note: n-grams have fixed overlap of $n-1$ symbols

Tries

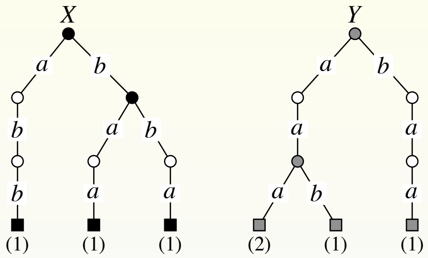
Efficient data structure for storage of sequences



- "Trie" = Re**trie**val tree (also dictionary or keyword tree)
- Path from root to marked node represents stored sequence
- Example sequences: {"ast", "bau", "baum", "beil"}

Implementation of N-grams

Efficient realization using Trie representation



Tries of 3-grams
for $x =$"abbaa"
and $y =$"baaaab"

- Kernel computation via parallel traversal of matching nodes
- Run-time $\mathcal{O}(n \cdot \min(|x|, |y|))$ for n-grams
- Blended n-grams by storing $\#_w(x)$ in inner nodes.

Positional N-grams

Incorporation of positional information into *n*-gram concept

x = "Hasta la vista, baby." $\rightarrow$ { "$H_1a_2s_3$", "$a_2s_3t_4$", "$s_5t_6a_7$", ... }

Weighted Degree Kernel

Sequence kernel using *n*-grams and extended alphabet

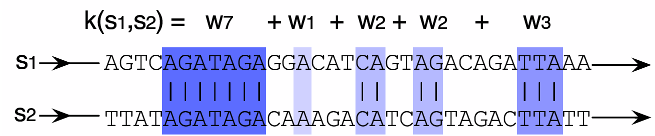
$$\tilde{\mathcal{A}} = \mathcal{A} \times \mathbb{N},$$

where for $(a, p) \in \tilde{\mathcal{A}}$, a encodes a symbol and p its position

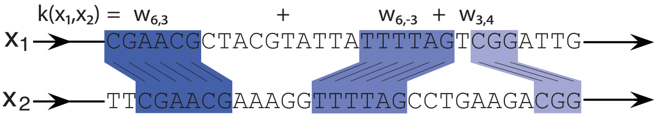
- Extension by incorporating minor positional shifts

Implementation of Positional N-grams

Efficient realization by looping over sequences



Implementation with shifts via multiple looping



Run-time $\mathcal{O}(s \cdot \max(|x|, |y|))$ with shift s

Contiguous Subsequences

Characterization using all possible subsequences

$$x = \text{“Hasta la vista, baby.”} \longrightarrow \{ \text{“H”}, \text{“Ha”}, \text{“Has”}, \ldots \}$$

Contiguous Subsequence Kernel

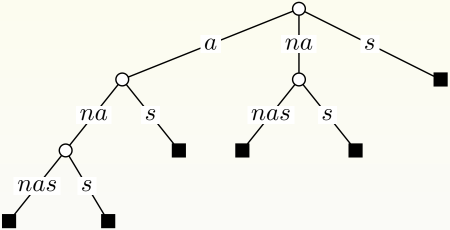
Sequence kernel using embedding language containing all possible sequences

$$L = \mathcal{A}^* = \bigcup_{i=1}^{\infty} \mathcal{A}^i$$

- Arbitary overlap $\Rightarrow$ quadratic amount of subsequences
- Weighting to control contribution of subsequences, e.g. length-dependent $N_w = \lambda^{|-w|}$ with $0 < \lambda \leq 1$

Suffix Trees

- Efficient and versatile sequence representation
- Suffix Tree = Trie containing all suffixes of a sequence

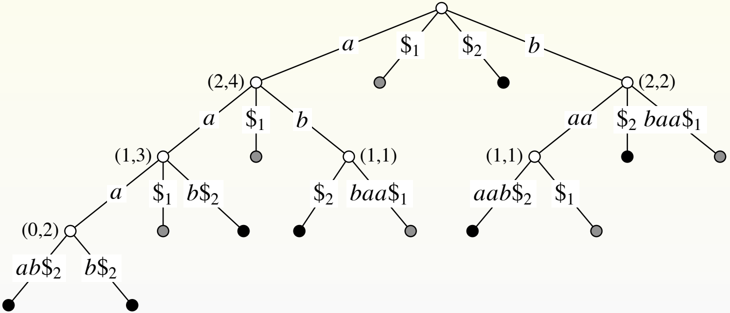


- Compact storage by edge compression, e.g. *nas* $\Rightarrow [4, 6]$
- Example sequence: "ananas"

Implementation of Subsequences

Efficient realization using generalized suffix trees (GST)

GST for x =“abbaa” and y =“baaaab” using z =“abaa$\$_1$baaab$\$_2$”



- Kernel computation via depth-first search in suffix tree
- $\mathcal{O}(|z|)$ inner nodes $\Rightarrow$ run-time $\mathcal{O}(|z|) = \mathcal{O}(|x| + |y|)$

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

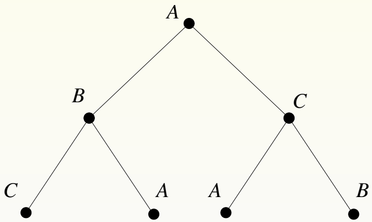
A tree x derived from a context-free grammar, such that each node $v \in V$ is associated with a production rule $p(v)$.

Further notation

- v_i = i-th child of node $v \in V$,
- $|v|$ = number of children of $v \in V$
- and the set $\mathcal{T}$ of all parse trees

Parse Trees

Tree representation of "sentences" derived from a grammar



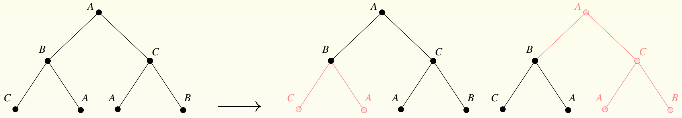
Parse tree for $CAAB$ using grammar over $\{A, B, C\}$ and productions

- $p_1 : A \rightarrow B\ C$
- $p_2 : B \rightarrow C\ A$
- $p_3 : C \rightarrow A\ B$

Common data structure in natural language processing and design of programming languages, compilers, etc.

Embedding subtrees

Characterization of parse trees by contained subtrees



Feature map

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

$$x \mapsto (\mathbb{I}_t(x))_{t \in \mathcal{T}}$$

where $\mathbb{I}_t(x)$ indicates if t is a subtree of parse tree x.

- Binary feature space spanned by indicator for subtrees

Parse Tree Kernel

Parse Tree Kernel

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

$$k(x, y) = \langle \phi(x), \phi(y) \rangle = \sum_{t \in T} \mathbb{I}_t(x)\mathbb{I}_t(y)$$

Proof.

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

- $\mathcal{T}$ has infinite size $\Rightarrow$ naive computation infeasible

Counting shared subtrees

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

$$k(x, y) = \sum_{t \in T} \mathbb{I}_t(x)\mathbb{I}_t(y) = \sum_{v \in V_x} \sum_{w \in V_y} c(v, w)$$

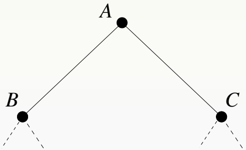
Counting function

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

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

Counting function in Detail

- First base case: $c(v,w) = 0$ if $p(v) \neq p(w)$
  $\Rightarrow$ trivial, no match = no shared subtrees
- Second base case: $c(v,w) = 1$ if $|v| = |w| = 0$
  $\Rightarrow$ trivial, one leave = one subtree
- Recursion: $c(v,w) = \prod_{i=1}^{|v|} (1 + c(v_i, w_i))$

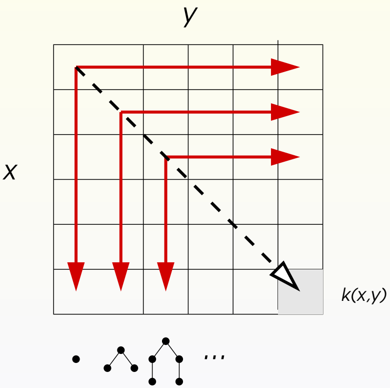


$$c(v_A, w_A) = (1 + c(v_B, w_B)) \cdot (1 + c(v_C, w_C))$$

Pair all shared subtrees in B with C including edges to A.

Implementation of Parse Tree Kernel

Realization using dynamic programming table.



Matrix of all $c(v, w)$ with $(v, w) \in V_x \times V_y$ ordered by descending depth

Run-time $\mathcal{O}(|V_x| \cdot |V_y|)$. Speed-up by skipping non-matching node pairs.

Shallow Tree Kernel

Idea: map trees to sequences and apply sequence kernels

Flattening

A function $f : \mathcal{T} \to \mathcal{A}^*$ mapping trees to sequences given by $f(x) \mapsto m(v^*)$ with

$$m(v) = \text{“[”} \circ m(v_1) \circ \cdots m(v_{|v|}) \circ \text{“]”}$$

Shallow tree kernel

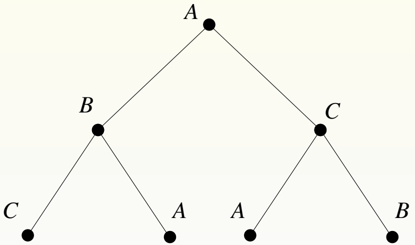
A kernel $k : \mathcal{T} \times \mathcal{T} \to \mathbb{R}$ based on a sequence kernel $\hat{k}$ given by

$$k(x, y) = \hat{k}(s(x), s(y)).$$

Flattening a Tree

Implementation of flattening

- Computation by depth-first traversal of tree
- Run-time dependent on traversal and sequence kernel



Example

- With labels $f(x) =$ “[A[B[C][A]][C[A][B]]]”
- Without labels $f(x) =$ “[[[][]][[][]]]”

Conclusions

Kernels for structured data

- Effective means for learning with structured data
- Various efficient kernels for sequences and trees

More on structured data and kernels

- Kernel for graphs, images, sounds
- . . .

Interesting applications (upcoming lectures)

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

References

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.

Sonnenburg, S., Rätsch, G., and Rieck, K. (2007).
Large scale learning with string kernels.
In Bottou, L., Chapelle, O., DeCoste, D., and Weston, J., editors, *Large Scale Kernel Machines*, pages 73–103. MIT Press, Cambridge, MA.