Kernel Functions for Structured Data

Dr. Konrad Rieck

Technische Universität Berlin

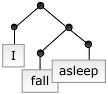
April 12, 2010

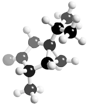
Outline

Structured Data

Structured data ubiquitous 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

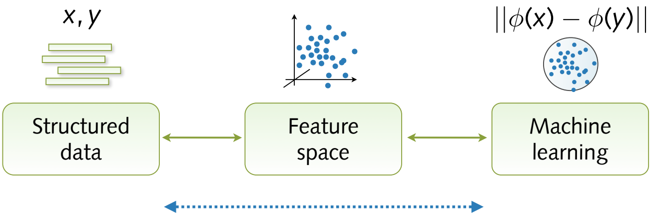
- $k(x, y) := \langle x, y \rangle = \sum_{i=1}^{d} x_i y_i$ (Linear kernel)
- $k(x, y) := (\langle x, y \rangle + \theta)^p$ (Polynomial kernel)
- $k(x, y) := \exp\left(\frac{||x-y||^2}{\gamma}\right)$ (Gaussian kernel)
- $k(x, y) := \tanh(\langle x, y \rangle + \theta)$ (Sigmoidal kernel)

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

Kernels and Structured 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



$$k(x, z) \mapsto \langle \phi(x), \phi(z) \rangle$$

Kernels for Strings

Strings

Alphabet

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

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

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. $\square$

Bag-of-Words

Characterization of strings using non-overlapping substrings

$x =$ `mary had a little lamb`

`mary` `had` `a` `little` `lamb`

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

$x =$ `mary had a little lamb`

`mar` `ary` `ry_` `y_h` `_ha` ...

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

$x =$ `mary had a little lamb`

`m` `ma` `mar` ... `a` `ar` ...

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

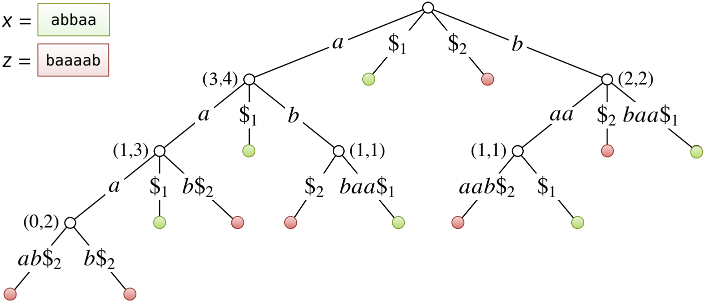
- Explicit kernel computation $\longrightarrow$ parallel loop over arrays
- Run-time $\mathcal{O}(|x| + |z|)$ for words with no or bounded overlap

Suffix Trees

Strings jointly stored in generalized suffix tree

$x =$ abbaa

$z =$ baaaab



- 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

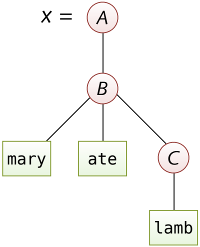
A tree x deriving from agrammar, such that each node $v \in V$ is associated with a production rule $p(v)$.

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

$x =$ 

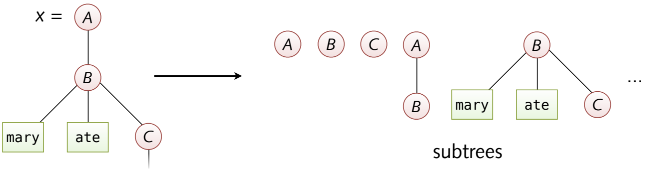
Parse tree for "mary ate lamb" with production rules

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

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

Embedding Trees

Characterization of parse trees using contained subtrees



Feature map

A function $\phi : T \rightarrow \mathbb{R}^{|T|}$ 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 $k : T \times T \longrightarrow \mathbb{R}$ 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. $\square$

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.

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

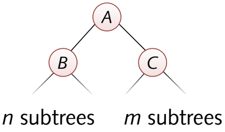
Counting function

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

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

Counting in Detail

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



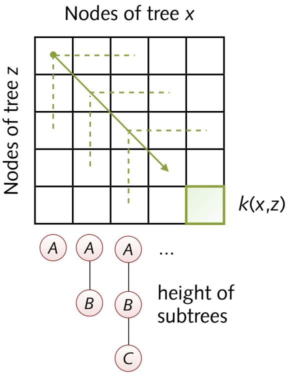
Count all combinations of shared subtrees below node A

$$c(v_A, w_A) = (n+1) \cdot (m+1)$$

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

Conclusions

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

References

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.