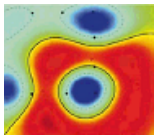
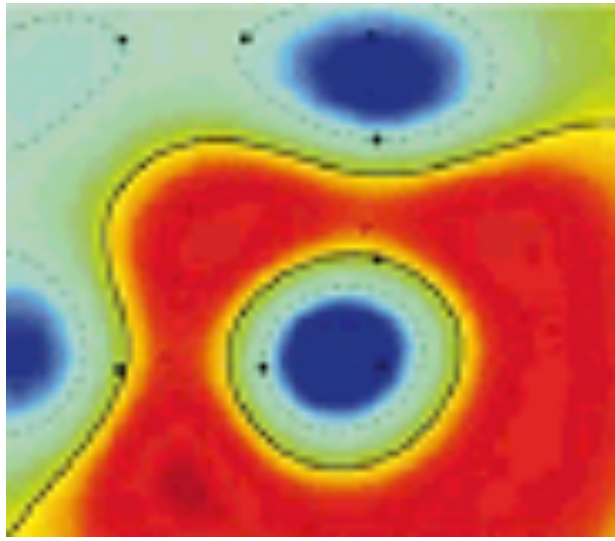


SoSe 2024

Machine Learning 2/2-X

Technische Universität Berlin



Lecture 6

Kernel Methods for Structured Inputs

Recap: From Linear to Kernel Models

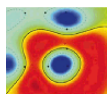
Linear models are very common in machine learning. They can solve a number of tasks (e.g. regression, classification, component analysis) and often come with desirable properties such as convexity, or closed-form solutions.

Example: Linear Regression

Let $X = [x_1 | \dots | x_N]$ be the training data, and $y = (y_1, \dots, y_N)$ be the targets. The linear regression model

$$f(x) = \beta^\top x$$

trained to minimize the least square error admits the closed-form solution $\beta = (XX^\top)^{-1}Xy$.



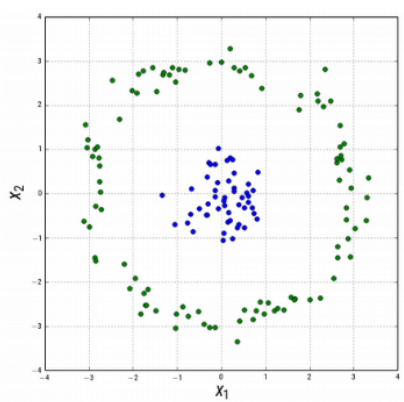
Recap: From Linear to Kernel Models

Idea

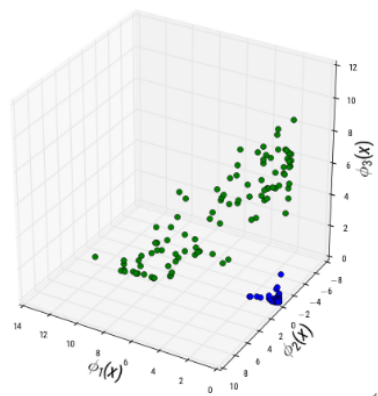
To make the model nonlinear (and be able to solve nonlinear problems), pass the data through some nonlinear feature map before applying the linear model

$$f(x) = \beta^\top \phi(x)$$

When the feature map is higher-dimensional, this increases the expressive power of the model.



$$\phi(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2, x_1, x_2)$$



Recap: From Linear to Kernel Models

Idea

Passing the data through some nonlinear feature map before applying the linear model

$$f(x) = \beta^\top \phi(x)$$

Express the model in the span of the data

$$\beta = \sum_{i=1}^{N} \alpha_i \phi(x_i)$$

The model and training algorithms can often be written in a way that involves the feature map *only* through dot products:

$$f(x) = \sum_{i=1}^{N} \alpha_i \phi(x_i)^\top \phi(x) \qquad \text{(model)}$$

$$\partial \mathcal{E} / \partial \alpha_i = \sum_{n=1}^{N} 2 \left[f(x_n) - y_n \right] \cdot \phi(x_i)^\top \phi(x_n) \qquad \text{(learning signal)}$$

Recap: From Linear to Kernel Models

Observation

Some dot products are expressed more compactly using general nonlinear functions, e.g.

$$\Big\langle \underbrace{(x_1^2, \sqrt{2}x_1x_2, x_2^2)}_{\phi(x)}, \underbrace{(y_1^2, \sqrt{2}y_1y_2, y_2^2)}_{\phi(y)} \Big\rangle = \underbrace{\big\langle (x_1, x_2), (y_1, y_2) \big\rangle^2}_{k(x,y)}$$

Question

Is the reverse true (i.e. does any kernel induces a feature map)?

$$k(x, x') \rightsquigarrow \phi(x)^\top \phi(x')$$

(needed for guaranteeing convergence and other regularization properties of the learning algorithm.)

Recap: Positive Semi-Definite Kernels

When the function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is

- symmetric, i.e. $k(x, x') = k(x', x)$, and
- positive semi-definite, i.e.

$$\sum_{i=1}^{N} \sum_{j=1}^{N} c_i c_j k(x_i, x_j) \geq 0$$

for all $x_1, \ldots, x_N \in \mathbb{R}^d$ and scalars $c_1, \ldots, c_N \in \mathbb{R}$,

it can be shown (Mercer Theorem) that there exists a feature map $\phi(x)$, potentially infinite-dimensional, such that:

$$k(x, x') = \phi(x)^\top \phi(x').$$

Recap: Examples of PSD Kernels

Example of kernels that can be generally apply to data in $\mathbb{R}^d$.

Linear kernel	$k(x, x') = x^\top x'$
Polynomial kernel	$k(x, x') = (x^\top x' + a)^b$
Gaussian kernel	$k(x, x') = \exp(-\gamma \cdot \|x - x'\|^2)$
t-Student kernel	$k(x, x') = (a + \|x - x'\|^2)^{-1}$

Question

Can kernels be built for *other type of data* (e.g. text, images, sequences, trees, graphs) and can they incorporate *domain-specific knowledge*?

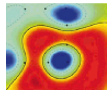
Outline of Today's Lecture

In this lecture, we will study a variety of structured kernels:

- String Kernels
  - Application in Genomics
- Tree Kernels
- Fisher Kernel
- Diffusion Kernels

Review paper

T. Gärtner. A Survey of Kernels for Structured Data, ACM SIGKDD Explorations Newsletter, 2003.



Representing Strings

Alphabet

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

- Set of states, $\mathcal{A} = \{S_1, \ldots, S_d\}$
- DNA, $\mathcal{A} = \{\mathrm{G}, \mathrm{A}, \mathrm{T}, \mathrm{C}\}$
- Natural language text,
  $\mathcal{A} = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \ldots, \mathrm{A}, \mathrm{B}, \mathrm{C}, \ldots .0, 1, 2, \ldots\}$

String or Sequence

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

- $\mathcal{A}^L$ = all strings of length L
- $\mathcal{A}^\star$ = all strings of any length

Designing a String Kernel

Assume two sequences $x = (\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{c}, \mathrm{e}, \mathrm{a}, \mathrm{b}, \mathrm{d})$, and $z = (\mathrm{a}, \mathrm{b}, \mathrm{a}, \mathrm{c}, \mathrm{c}, \mathrm{c}, \mathrm{b}, \mathrm{e})$. How to compute a kernel score $k(x, z)$ to compare these two sequences?

Approach 1

Count the number of matching symbols.

x	a	b	c	c	e	a	b	d
z	a	b	a	c	c	c	b	e
$I(x_i = z_i)$	1	1	0	1	0	0	1	0

$$k(x, z) = \sum_{i=1}^{L} I(x_i = z_i)$$

String Kernel and Feature Map

Proposition: The kernel $k\colon \mathcal{A}^L \times \mathcal{A}^L \to \mathbb{R}$ that counts the number of matching symbols between the two input sequences, i.e.

$$k(x, z) = \sum_{i=1}^{L} I(x_i = z_i)$$

induces the feature map

$$\phi(x) = \big(I(x_i = a)\big)_{a \in \mathcal{A}, i \in \{1 \ldots L\}}$$

i.e. an array of indicator variables, of size $L \times |\mathcal{A}|$.

Exercise: Prove this.

Designing a String Kernel

Approach 2

Count the number of matching symbols *with some shift tolerance*.

x	a	b	c	c	e	a	b	d
z	a	b	a	c	c	c	b	e
$I(x_i = z_{i-1})$	-	0	0	0	0	0	0	0
$I(x_i = z_i)$	1	1	0	1	0	0	1	0
$I(x_i = z_{i+1})$	0	0	1	1	0	0	0	-

$$k(x,z) = \sum_{i=1}^{L} \alpha \cdot I(x_i = z_{i-1}) + \beta \cdot I(x_i = z_i) + \gamma \cdot I(x_i = z_{i+1})$$

Exercise: Show that this function is a PSD kernel when setting $\alpha = 1, \beta = 2, \gamma = 1$, but it is not the case when setting $\alpha, \beta, \gamma = 1$.

Designing a String Kernel

Approach 3

Count the number of matching symbols between sequences $x = (\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{c}, \mathrm{e}, \mathrm{a}, \mathrm{b}, \mathrm{d})$, and $z = (\mathrm{a}, \mathrm{b}, \mathrm{a}, \mathrm{c}, \mathrm{c}, \mathrm{c}, \mathrm{b}, \mathrm{e})$ *with full shift invariance*.

x	aa	bb	cc	d	e
z	aa	bb	ccc		e
	4	4	6	0	1

This gives the 'Bag-of-Words' kernel:

$$k(x, z) = \sum_{i=1}^{L} \sum_{j=1}^{L} I(x_i = z_j)$$
$$= \sum_{w} \#_w(x) \cdot \#_w(z)$$

Designing a String Kernel

Approach 4

Count the number of matching between sequences $x = (\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{c}, \mathsf{e}, \mathsf{a}, \mathsf{b}, \mathsf{d})$, and $z = (\mathsf{a}, \mathsf{b}, \mathsf{a}, \mathsf{c}, \mathsf{c}, \mathsf{c}, \mathsf{b}, \mathsf{e})$ of *subsequences of* symbols.

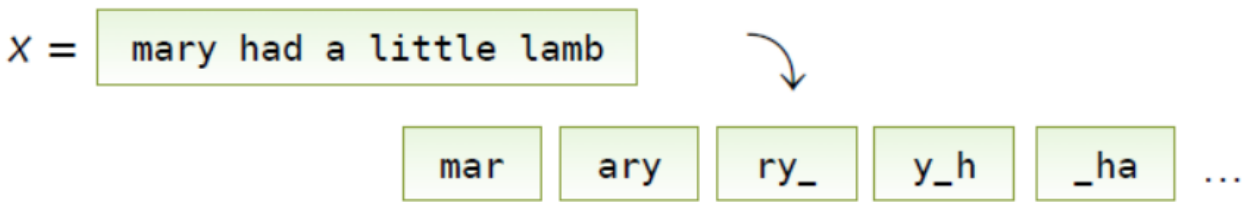
x	ab	bc	cc	ce	ea	ab	bd
z	ab	ba	ac	cc	cc	cb	be
$I(x_{i,i+1} = z_{i,i+1})$	1	0	0	0	0	0	0

Can be generalized to triplets of consecutive of symbols, or more generally, 'n-grams':

$$k(x, z) = \sum_{i=1}^{L+1-n} I(x_{i\ldots i+n-1} = z_{i\ldots i+n-1})$$

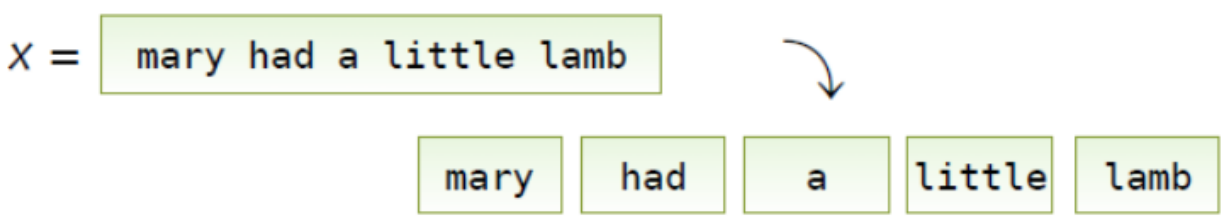
N-Gram vs. Tokenization

N-Gram Approach



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

Tokenization approach



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

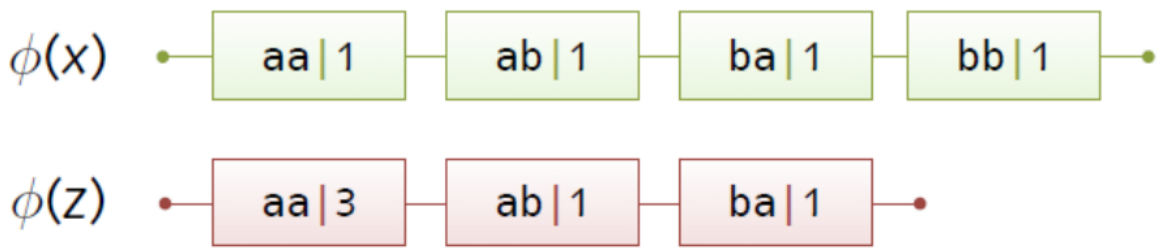
Implementing String Kernels Efficiently

Example: Bag-of-Words Kernel

$$k(x, z) = \sum_w \#_w(x) \cdot \#_w(z)$$

Implementation trick

Leverage the fact that many kernel evaluations are needed (the whole kernel matrix $K = (k(x_i, x_j))_{i,j=1}^N$), e.g. build sorted lists before performing the kernel evaluations.



Application: Genomic Data

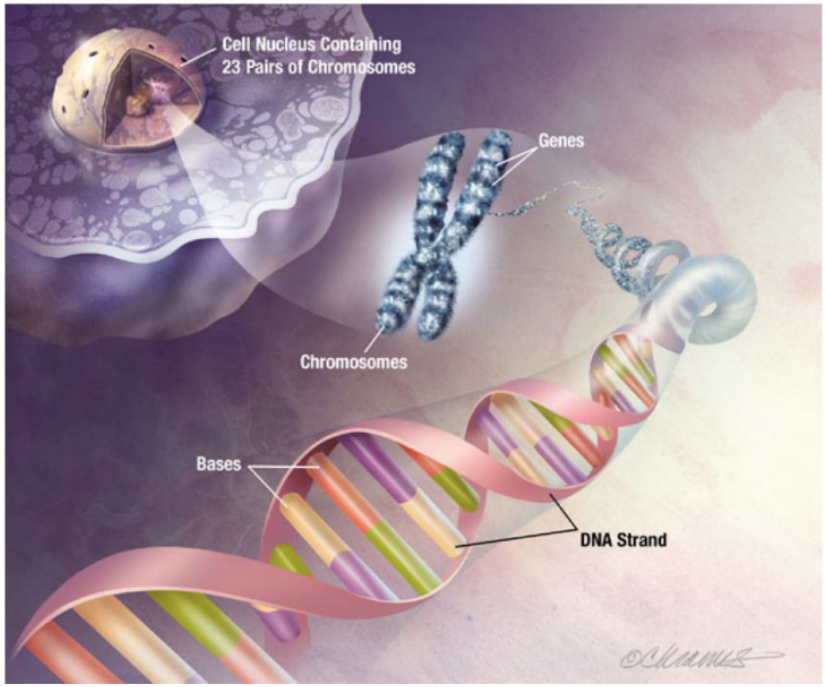
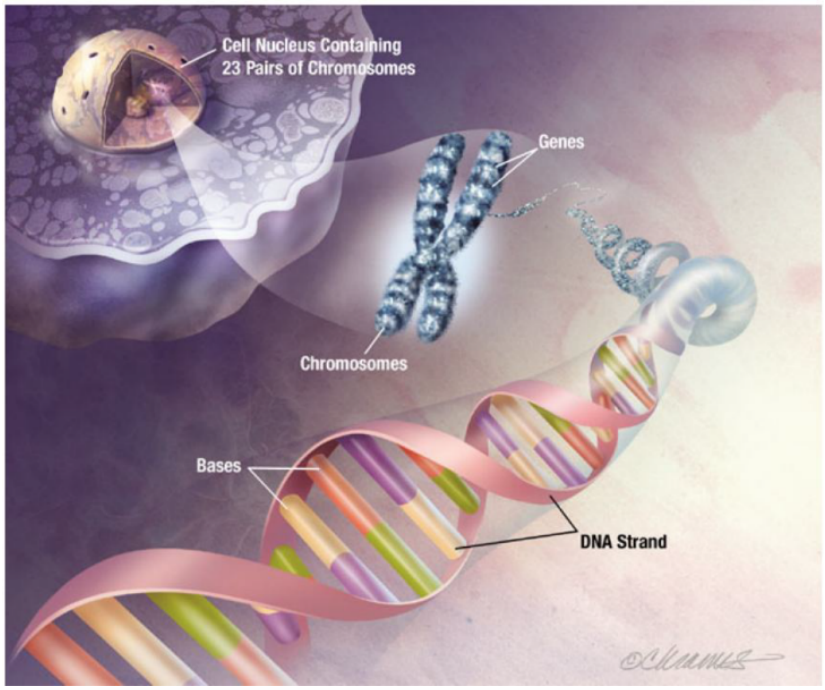


Figure taken from
http://education.technyou.edu.au/view/91/155/what-does-dna-look

Organism's complete set of DNA contains all information needed to build and maintain an organism.

Humans genome consists of more than 3 billion DNA base pairs.

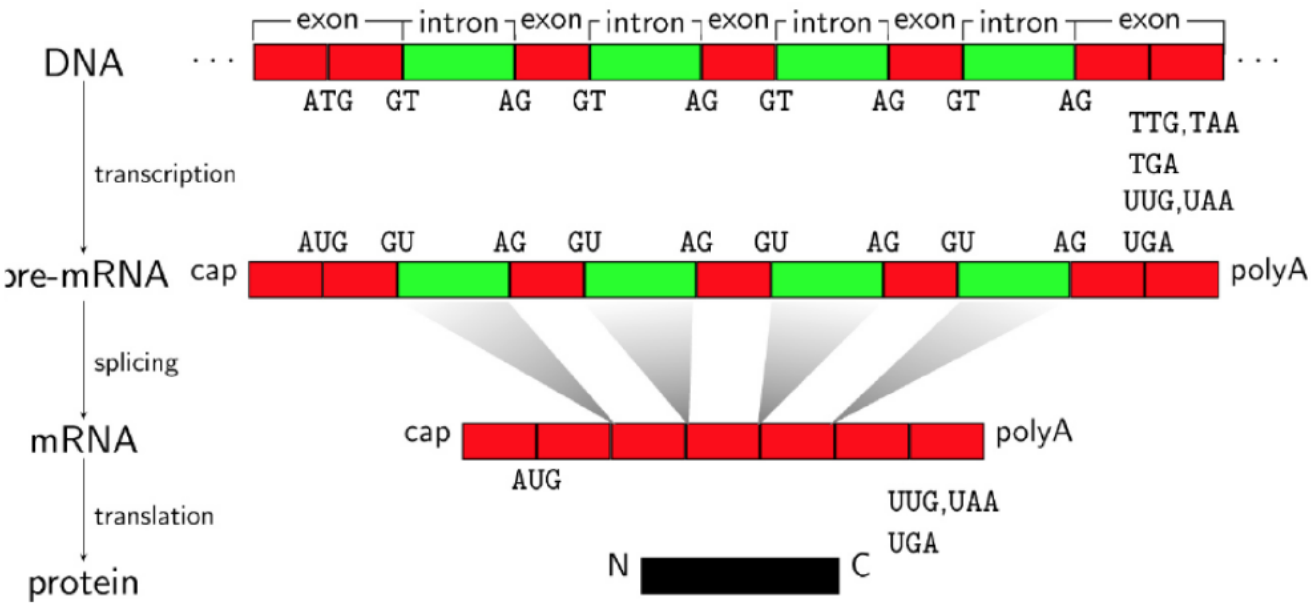
From Genes to Proteins



- A gene is a segment of DNA that codes for a certain property (protein).
- Proteins control enzymes (catalyzed; involved in metabolism, DNA replication/repair, RNA synthesis, ...), cell signaling (Insulin), ligand binding (Haemoglobin), ...

From Genes to Proteins

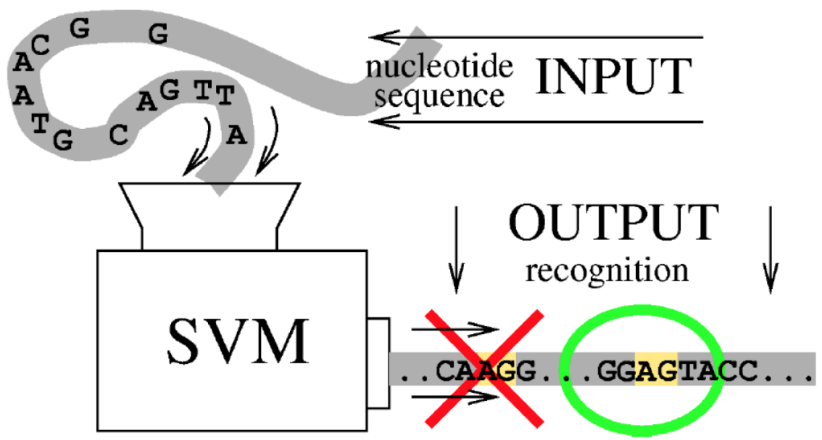
Protein biosynthesis: process from genes to proteins



In the splicing step, the introns are removed and the exons are joined together at the splice sites.

Splice Site Prediction

Idea: Build a machine learning model (kernel SVM) that predicts splice sites (transition from exon to intron). The kernel SVM is then applied as a sliding window through large nucleotide sequences.



Paper

S. Sonnenburg et al. Accurate splice site prediction using support vector machines, BMC Bioinformatics, 2007

Weighted Degree Kernel

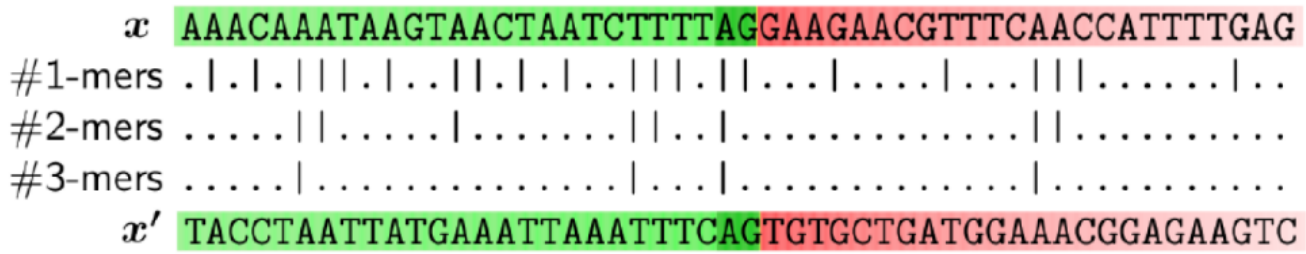
- The Weighted Degree Kernel is used to detect if a splice site is currently at the center of the input window.
- Weighted sum of n-gram kernels of maximum length d.
- The kernel receives as input two nucleotide sequences of size L (i.e. $\in \{\mathsf{G}, \mathsf{A}, \mathsf{T}, \mathsf{C}\}^L$) and computes

$$k(x, z) = \sum_{n=1}^{d} \beta_n \sum_{i=1}^{L+1-n} I(x_{i \ldots i+n-1} = z_{i \ldots i+n-1})$$

where $x_{i \ldots i+n-1}$ is a subsequence starting at index i and of length n.

Weighted Degree Kernel

Example: Weighted degree kernel with parameter $d = 3$ applied to two input sequences of length $L = 50$.



Computational trick

Weighted degree kernel can be computed quickly by identifying large blocks.

Kernel for 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

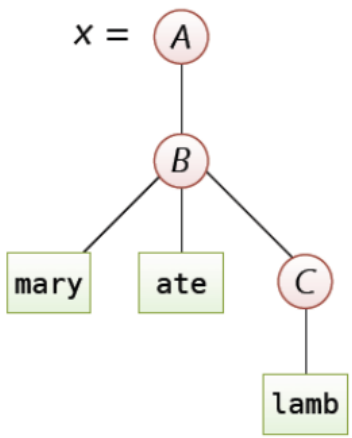
- v_i = i-th child of node $v \in V$
- $|v|$ = number of children of $v \in V$
- T = set of all possible parse trees

Reference paper:

K Rieck et al. Approximate Tree Kernels. J. Mach. Learn. Res. 11: 555-580 (2010)

Parse Tree

Tree representation of "sentences" derived from a grammar



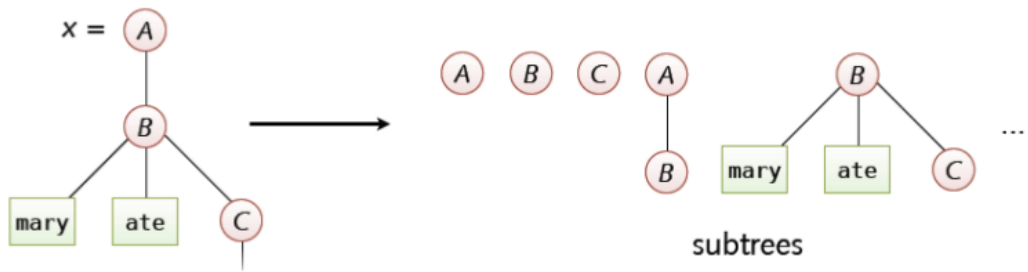
Parse tree for "`mary ate lamb`" with production rules

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

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

Parse Tree Features

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

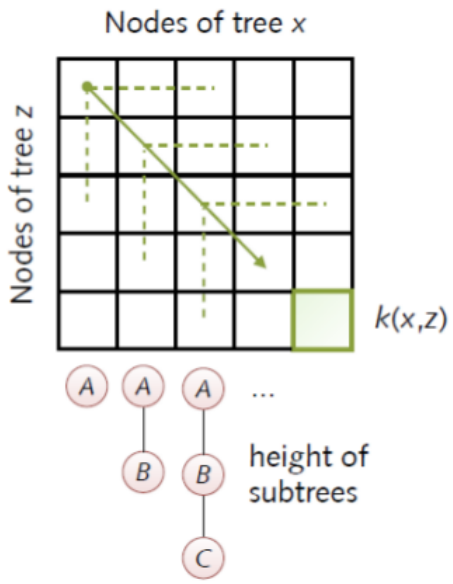
- $c(v,w) = 0$ if $p(v) \neq p(w)$ (different production)
- $c(v,w) = 1$ if $|v| = |w| = 0$ (leaf nodes)
- otherwise

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

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

Model-Induced Kernels

Observation

Certain machine learning models already exist for structured data, e.g. hidden Markov models, graph neural networks.

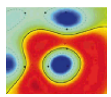
Idea: Do not build a new kernel directly on the input data, extract a kernel from local response of the existing model $f_\theta(x)$ to its parameter vector $\theta \in \mathbb{R}^{|\theta|}$, e.g. define $\psi \colon \mathcal{X} \to \mathbb{R}^{|\theta|}$.

$$\psi(x) = \nabla_\theta f_\theta(x)$$

and construct the kernel as:

$$k(x, x') = \kappa(\psi(x), \psi(x'))$$

where κ is a standard kernel (e.g. linear, etc.).



Example: The Fisher Kernel

Suppose $x \in \mathcal{X}$ and we have some generative model for the data

$$p_\theta(x)$$

parameterized by some vector $\theta \in \mathbb{R}^h$ learned from the data. An example of generative model for structured data is the Hidden Markov Model.

The Fisher Kernel is built as follows. It computes the gradient of the locally evaluated function w.r.t. the model parameters.

$$G_x = \frac{\partial}{\partial \theta} \log p_\theta(x)$$

Then, the kernel is built as:

$$k(x, x') = G_x^\top \left(\mathbb{E}_{z \sim p_\theta} [G_z G_z^\top] \right)^{-1} G_{x'}$$

Diffusion Kernels

Diffusion kernels assume that the input domain is discrete and that the local geometry is given by a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where each node represents a data point. The diffusion kernel defines the *generator matrix*:

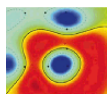
$$H_{ij} = \begin{cases} 1 & (\nu_i, \nu_j) \in \mathcal{E} \\ -\deg(\nu_i) & \nu_i = \nu_j \\ 0 & \text{otherwise} \end{cases}$$

and then *diffuses* the graph signal by matrix exponentiation. Kernel scores are then given by:

$$k(x_i, x_j) = [\, e^{\beta H} \,]_{ij}$$

with β a kernel hyperparameter.

Application: The diffusion kernel is particularly useful for semi-supervised learning, or for unsupervised analyses such as clustering (cf. spectral clustering).



Example: Diffusion Kernel on 2D Data

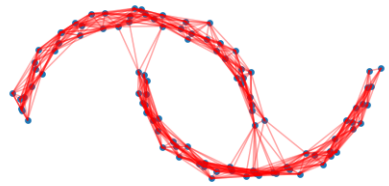
1. Build a dataset (e.g. with manifold structure).

2. Build a graph connecting the neighboring data points.

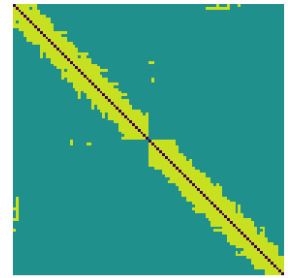
3. Build the generator matrix H on which the diffusion kernel is based.

Dataset

Graph



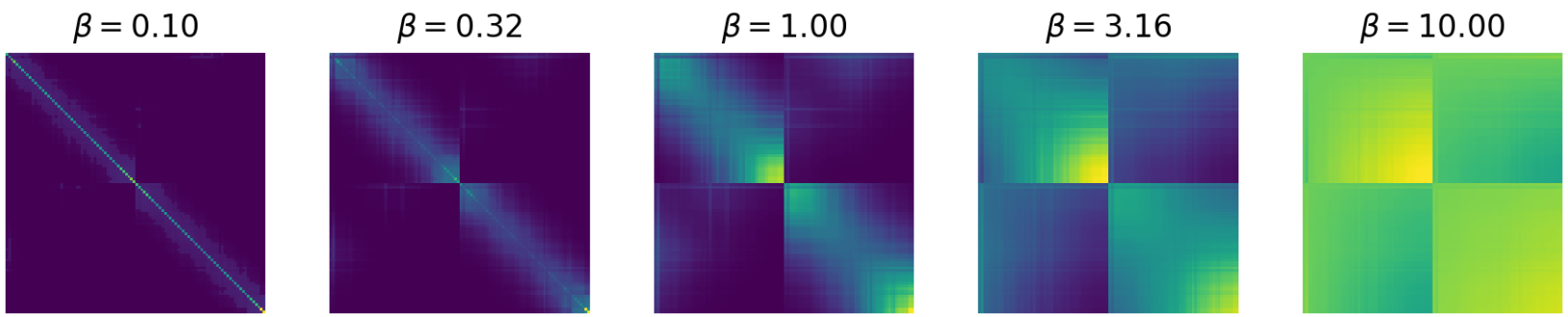
Generator matrix H



Example: Diffusion Kernel on 2D Data

Results

Kernel scores $K = \exp(\beta H)$ with different diffusion parameters β



- The higher the parameter β, the higher the similarity between points of the same cluster.
- A too high parameter β starts leaking though the clusters and produce high similarity scores everywhere.

Summary

- Structured kernels can be used to predict data which is not in $\mathbb{R}^d$, e.g. sequences of symbols or trees.
- Structured kernels can be designed to incorporate prior knowledge, e.g. positional invariance, diffusion on a graph.
- Once the structured kernel has been defined, most popular learning algorithms, e.g. least square regression, SVMs, PCA, CCA, etc. can be used.