Support vector machines and applications in computational biology

Jean-Philippe Vert
Jean-Philippe.Vert@mines.org

MINES ParisTech
institut Curie
Together, let's beat cancer.
Outline

1. Motivations
2. Linear SVM
3. Nonlinear SVM and kernels
4. Kernels for strings and graphs
Outline

1. Motivations
2. Linear SVM
3. Nonlinear SVM and kernels
4. Kernels for strings and graphs
Problem 1

Given the expression levels of 20k genes in a leukemia, is it an acute lymphocytic or myeloid leukemia (ALL or AML)?
Problem 2

Given the expression levels of 20k genes in a tumour after surgery, is it likely to relapse later?
Pharmacogenomics / Toxicogenomics

Problem 3
Given the genome of a person, which drug should we give?
Protein annotation

Data available

- **Secreted proteins:**
  - MASKATLLLAAFATLLLFATCIARHQQRQQQQQCQLQNIEA...
  - MARSSLFTFLCLAVFINGCLSQIEQQSPWEFQGSEVW...
  - MALHTVILIMLSLLPMLEAQNPEHANITIGEPITNETLGWL...
  - ...

- **Non-secreted proteins:**
  - MAPPSVFAEVPQAQPVLVFKLIAFDREDPDPRKVNLGVG...
  - MAHTLGLTQPNSTEPHKISFTAKEIDVIEWKGDLVVG...
  - MSISESYAKEIKTAFRQFTDFPIEGEQFEDFLPIIGNP..
  - ...

Problem 4

Given a newly sequenced protein, is it secreted or not?
Problem 5
Given a new candidate molecule, is it likely to be active?
A common topic
A common topic
A common topic
A common topic
On real data...
Pattern recognition, *aka* supervised classification

- **Challenges**
  - High dimension
  - Few samples
  - Structured data
  - Heterogeneous data
  - Prior knowledge
  - Fast and scalable implementations
  - Interpretable models
1. Motivations
2. Linear SVM
3. Nonlinear SVM and kernels
4. Kernels for strings and graphs
Linear classifier
Linear classifier
Linear classifier
Linear classifier
Linear classifier
Linear classifier
Linear classifier
Linear classifier
Which one is better?
The margin of a linear classifier
The margin of a linear classifier
The margin of a linear classifier
The margin of a linear classifier
The margin of a linear classifier
Largest margin classifier (*hard-margin SVM*)
Support vectors
More formally

The training set is a finite set of $n$ data/class pairs:

$$S = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)\} ,$$

where $\vec{x}_i \in \mathbb{R}^p$ and $y_i \in \{-1, 1\}$.

We assume (for the moment) that the data are linearly separable, i.e., that there exists $(\vec{w}, b) \in \mathbb{R}^p \times \mathbb{R}$ such that:

$$\begin{cases} 
\vec{w} \cdot \vec{x}_i + b > 0 & \text{if } y_i = 1 , \\
\vec{w} \cdot \vec{x}_i + b < 0 & \text{if } y_i = -1 .
\end{cases}$$
How to find the largest separating hyperplane?

For a given linear classifier $f(x) = \vec{w} \cdot \vec{x} + b$ consider the "tube" defined by the values $-1$ and $+1$ of the decision function:
Indeed, the points $\vec{x}_1$ and $\vec{x}_2$ satisfy:

$$\begin{cases} 
\vec{w} \cdot \vec{x}_1 + b = 0, \\
\vec{w} \cdot \vec{x}_2 + b = 1.
\end{cases}$$

By subtracting we get $\vec{w} . (\vec{x}_2 - \vec{x}_1) = 1$, and therefore:

$$\gamma = 2\|\vec{x}_2 - \vec{x}_1\| = \frac{2}{\|\vec{w}\|}.$$
All training points should be on the correct side of the dotted line.

For positive examples \((y_i = 1)\) this means:

\[
\vec{w} \cdot \vec{x}_i + b \geq 1.
\]

For negative examples \((y_i = -1)\) this means:

\[
\vec{w} \cdot \vec{x}_i + b \leq -1.
\]

Both cases are summarized by:

\[
\forall i = 1, \ldots, n, \quad y_i \left( \vec{w} \cdot \vec{x}_i + b \right) \geq 1.
\]
Finding the optimal hyperplane

Find \((\vec{w}, b)\) which minimize:

\[ \| \vec{w} \|^2 \]

under the constraints:

\[ \forall i = 1, \ldots, n, \quad y_i (\vec{w} \cdot \vec{x}_i + b) - 1 \geq 0. \]

This is a classical quadratic program on \(\mathbb{R}^{p+1}\).
In order to minimize:

\[ \frac{1}{2} \| \vec{w} \|^2 \]

under the constraints:

\[ \forall i = 1, \ldots, n, \quad y_i (\vec{w} \cdot \vec{x}_i + b) - 1 \geq 0, \]

we introduce one dual variable \( \alpha_i \) for each constraint, i.e., for each training point. The Lagrangian is:

\[
L (\vec{w}, b, \vec{\alpha}) = \frac{1}{2} \| \vec{w} \|^2 - \sum_{i=1}^{n} \alpha_i (y_i (\vec{w} \cdot \vec{x}_i + b) - 1) .
\]
Lagrangian

- $L(\vec{w}, b, \vec{\alpha})$ is convex quadratic in $\vec{w}$. It is minimized for:

$$\nabla_{\vec{w}} L = \vec{w} - \sum_{i=1}^{n} \alpha_i y_i \vec{x}_i = 0 \quad \implies \quad \vec{w} = \sum_{i=1}^{n} \alpha_i y_i \vec{x}_i.$$

- $L(\vec{w}, b, \vec{\alpha})$ is affine in $b$. Its minimum is $-\infty$ except if:

$$\nabla_b L = \sum_{i=1}^{n} \alpha_i y_i = 0.$$
We therefore obtain the Lagrange dual function:

\[
q(\vec{\alpha}) = \inf_{\vec{w} \in \mathbb{R}^p, b \in \mathbb{R}} L (\vec{w}, b, \vec{\alpha})
\]

\[
= \begin{cases} 
\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \vec{x}_i \cdot \vec{x}_j & \text{if } \sum_{i=1}^{n} \alpha_i y_i = 0, \\
-\infty & \text{otherwise.}
\end{cases}
\]

The dual problem is:

maximize \( q(\vec{\alpha}) \)

subject to \( \vec{\alpha} \geq 0 \).
Find $\alpha^* \in \mathbb{R}^n$ which maximizes

$$L(\vec{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j,$$

under the (simple) constraints $\alpha_i \geq 0$ (for $i = 1, \ldots, n$), and

$$\sum_{i=1}^{n} \alpha_i y_i = 0.$$

This is a quadratic program on $\mathbb{R}^N$, with "box constraints". $\vec{\alpha}^*$ can be found efficiently using dedicated optimization softwares.
Once $\bar{\alpha}^*$ is found, we recover $(\bar{w}^*, b^*)$ corresponding to the optimal hyperplane. $w^*$ is given by:

$$\bar{w}^* = \sum_{i=1}^{n} \alpha_i \bar{x}_i,$$

and the decision function is therefore:

$$f^*(\bar{x}) = \bar{w}^* \cdot \bar{x} + b^*$$

$$= \sum_{i=1}^{n} \alpha_i \bar{x}_i \cdot \bar{x} + b^*. \quad (1)$$
Interpretation: support vectors

\[ \alpha > 0 \]

\[ \alpha = 0 \]
What if data are not linearly separable?
What if data are not linearly separable?
What if data are not linearly separable?
What if data are not linearly separable?
Soft-margin SVM

- Find a trade-off between **large margin** and **few errors**.
- Mathematically:

  $$
  \min_f \left\{ \frac{1}{\text{margin}(f)} + C \times \text{errors}(f) \right\}
  $$

- $C$ is a parameter
The margin of a labeled point \((\vec{x}, y)\) is

\[
\text{margin}(\vec{x}, y) = y (\vec{w} \cdot \vec{x} + b)
\]

The error is
- 0 if \(\text{margin}(\vec{x}, y) > 1\),
- \(1 - \text{margin}(\vec{x}, y)\) otherwise.

The soft margin SVM solves:

\[
\min_{\vec{w}, b} \left\{ \|\vec{w}\|^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i (\vec{w} \cdot \vec{x}_i + b)) \right\}
\]
Soft-margin SVM and hinge loss

\[ \min_{\vec{w},b} \left\{ \sum_{i=1}^{n} \ell_{\text{hinge}} (\vec{w}.x_i + b, y_i) + \lambda \| \vec{w} \|^2_2 \right\} , \]

for \( \lambda = 1/C \) and the hinge loss function:

\[ \ell_{\text{hinge}} (u, y) = \max (1 - yu, 0) = \begin{cases} 0 & \text{if } yu \geq 1, \\ 1 - yu & \text{otherwise.} \end{cases} \]
Maximize

\[ L(\vec{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j, \]

under the constraints:

\[
\begin{cases} 
0 \leq \alpha_i \leq C, & \text{for } i = 1, \ldots, n \\
\sum_{i=1}^{n} \alpha_i y_i = 0. 
\end{cases}
\]
Interpretation: bounded and unbounded support vectors

$\alpha = 0$

$0 < \alpha < C$

$\alpha = C$

$0 < \alpha < C$
Primal (for large $n$) vs dual (for large $p$) optimization

1. Find $(\vec{w}, b) \in \mathbb{R}^{p+1}$ which solve:

$$\min_{\vec{w}, b} \left\{ \sum_{i=1}^{n} \ell_{\text{hinge}}(\vec{w}.x_i + b, y_i) + \lambda \| \vec{w} \|_2^2 \right\}. $$

2. Find $\alpha^* \in \mathbb{R}^n$ which maximizes

$$L(\vec{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \vec{x}_i.\vec{x}_j,$$

under the constraints:

$$\begin{cases} 
0 \leq \alpha_i \leq C, & \text{for } i = 1, \ldots, n \\
\sum_{i=1}^{n} \alpha_i y_i = 0. 
\end{cases}$$
Outline

1. Motivations
2. Linear SVM
3. Nonlinear SVM and kernels
4. Kernels for strings and graphs
Sometimes linear methods are not interesting
For $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ let $\Phi(x) = \begin{pmatrix} x_1^2 \\ x_2^2 \end{pmatrix}$. The decision function is:

$$f(x) = x_1^2 + x_2^2 - R^2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}^\top \begin{pmatrix} x_1^2 \\ x_2^2 \end{pmatrix} - R^2 = \beta^\top \Phi(x) + b.$$
Kernel = inner product in the feature space

**Definition**

For a given mapping

\[ \Phi : \mathcal{X} \mapsto \mathcal{H} \]

from the space of objects \( \mathcal{X} \) to some Hilbert space of features \( \mathcal{H} \), the **kernel** between two objects \( x \) and \( x' \) is the inner product of their images in the features space:

\[ \forall x, x' \in \mathcal{X}, \quad K(x, x') = \Phi(x)^\top \Phi(x'). \]
Example

Let $\mathcal{X} = \mathcal{H} = \mathbb{R}^2$ and for $x = \left( \begin{array}{c} x_1 \\ x_2 \end{array} \right)$ let $\Phi(x) = \left( \begin{array}{c} x_1^2 \\ x_2^2 \end{array} \right)$

Then

$$K(x, x') = \Phi(x)^\top \Phi(x') = (x_1)^2(x_1')^2 + (x_2)^2(x_2')^2.$$
The kernel tricks

2 tricks

1. Many linear algorithms (in particular linear SVM) can be performed in the feature space of $\Phi(x)$ without explicitly computing the images $\Phi(x)$, but instead by computing kernels $K(x, x')$.

2. It is sometimes possible to easily compute kernels which correspond to complex large-dimensional feature spaces: $K(x, x')$ is often much simpler to compute than $\Phi(x)$ and $\Phi(x')$.
Trick 1 : SVM in the original space

- Train the SVM by maximizing

\[
\max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^\top x_j,
\]

under the constraints:

\[
\left\{
\begin{array}{l}
0 \leq \alpha_i \leq C, \quad \text{for } i = 1, \ldots, n \\
\sum_{i=1}^{n} \alpha_i y_i = 0.
\end{array}
\right.
\]

- Predict with the decision function

\[
f(x) = \sum_{i=1}^{n} \alpha_i y_i x_i^\top x + b^*.
\]
Trick 1 : SVM in the feature space

- Train the SVM by maximizing

\[
\max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \Phi(x_i)^\top \Phi(x_j),
\]

under the constraints:

\[
\begin{cases}
0 \leq \alpha_i \leq C, & \text{for } i = 1, \ldots, n \\
\sum_{i=1}^{n} \alpha_i y_i = 0.
\end{cases}
\]

- Predict with the decision function

\[
f(x) = \sum_{i=1}^{n} \alpha_i y_i \Phi(x_i)^\top \Phi(x) + b^* .
\]
Trick 1: SVM in the feature space with a kernel

- Train the SVM by maximizing

\[
\max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j),
\]

under the constraints:

\[
\begin{cases}
0 \leq \alpha_i \leq C, & \text{for } i = 1, \ldots, n \\
\sum_{i=1}^{n} \alpha_i y_i = 0.
\end{cases}
\]

- Predict with the decision function

\[
f(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x) + b^*.
\]
For $x = (x_1, x_2) \top \in \mathbb{R}^2$, let $\Phi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \in \mathbb{R}^3$:

$$K(x, x') = x_1^2 x_1'^2 + 2x_1 x_2 x_1' x_1' + x_2^2 x_2'^2$$

$$= (x_1 x_1' + x_2 x_2')^2$$

$$= (x \top x')^2.$$
More generally, for $x, x' \in \mathbb{R}^p$,

$$K(x, x') = \left( x^\top x' + 1 \right)^d$$

is an inner product in a feature space of all monomials of degree up to $d$ (left as exercise.)
Combining tricks: learn a polynomial discrimination rule with SVM

- Train the SVM by maximizing

\[
\max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i^\top x_j + 1)^d,
\]

under the constraints:

\[
\begin{cases}
0 \leq \alpha_i \leq C, & \text{for } i = 1, \ldots, n \\
\sum_{i=1}^{n} \alpha_i y_i = 0.
\end{cases}
\]

- Predict with the decision function

\[
f(x) = \sum_{i=1}^{n} \alpha_i y_i (x_i^\top x + 1)^d + b^*.
\]
Illustration: toy nonlinear problem

```r
> plot(x, col=ifelse(y>0,1,2), pch=ifelse(y>0,1,2))
```

Training data
Illustration: toy nonlinear problem, linear SVM

```r
> library(kernlab)
> svp <- ksvm(x, y, type="C-svc", kernel='vanilladot')
> plot(svp, data=x)
```
> svp <- ksvm(x,y,type="C-svc", ...
    kernel=polydot(degree=2))
> plot(svp, data=x)
Which functions $K(x, x')$ are kernels?

**Definition**

A function $K(x, x')$ defined on a set $\mathcal{X}$ is a kernel if and only if there exists a features space (Hilbert space) $\mathcal{H}$ and a mapping

$$\Phi : \mathcal{X} \mapsto \mathcal{H},$$

such that, for any $x, x'$ in $\mathcal{X}$:

$$K(x, x') = \langle \Phi(x), \Phi(x') \rangle_{\mathcal{H}}.$$
Definition

A positive definite (p.d.) function on the set $\mathcal{X}$ is a function $K: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ symmetric:

$$\forall (x, x') \in \mathcal{X}^2, \quad K(x, x') = K(x', x),$$

and which satisfies, for all $N \in \mathbb{N}$, $(x_1, x_2, \ldots, x_N) \in \mathcal{X}^N$ et $(a_1, a_2, \ldots, a_N) \in \mathbb{R}^N$:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j K(x_i, x_j) \geq 0.$$
Kernels are p.d. functions

Theorem (Aronszajn, 1950)

*K is a kernel if and only if* it is a positive definite function.
Proof?

Kernel $\Rightarrow$ p.d. function:
- $\langle \Phi(x), \Phi(x') \rangle_{\mathbb{R}^d} = \langle \Phi(x'), \Phi(x) \rangle_{\mathbb{R}^d}$,
- $\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j \langle \Phi(x_i), \Phi(x_j) \rangle_{\mathbb{R}^d} = \| \sum_{i=1}^{N} a_i \Phi(x_i) \|_{\mathbb{R}^d}^2 \geq 0$.

P.d. function $\Rightarrow$ kernel: more difficult...
Example: SVM with a Gaussian kernel

Training:

\[
\min_{\alpha \in \mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \exp \left( -\frac{||\vec{x}_i - \vec{x}_j||^2}{2\sigma^2} \right)
\]

s.t. \( 0 \leq \alpha_i \leq C \), and \( \sum_{i=1}^{n} \alpha_i y_i = 0 \).

Prediction

\[
f(\vec{x}) = \sum_{i=1}^{n} \alpha_i \exp \left( -\frac{||\vec{x} - \vec{x}_i||^2}{2\sigma^2} \right)
\]
Example: SVM with a Gaussian kernel

\[ f(\vec{x}) = \sum_{i=1}^{n} \alpha_i \exp \left( -\frac{||\vec{x} - \vec{x}_i||^2}{2\sigma^2} \right) \]
Linear vs nonlinear SVM
Regularity vs data fitting trade-off
$C$ controls the trade-off

$$\min_f \left\{ \frac{1}{\text{margin}(f)} + C \times \text{errors}(f) \right\}$$

- **Large $C$**: makes few errors
- **Small $C$**: ensure a large margin
- **Intermediate $C$**: finds a trade-off
Why it is important to control the trade-off
How to choose $C$ in practice

- Split your dataset in two ("train" and "test")
- Train SVM with different $C$ on the "train" set
- Compute the accuracy of the SVM on the "test" set
- Choose the $C$ which minimizes the "test" error
- (you may repeat this several times = cross-validation)
SVM summary

- Large margin
- Linear or nonlinear (with the kernel trick)
- Control of the regularization / data fitting trade-off with $C$
Outline

1. Motivations
2. Linear SVM
3. Nonlinear SVM and kernels
4. Kernels for strings and graphs
Supervised sequence classification

Data (training)
- Secreted proteins:
  - MASKATLLLAFTLLFATCIARHQQRQQQQNQCQLQNIEA...
  - MARSSLFTFLCLAVFINGCLSQIEQQSPWEFQGSEVW...
  - MALHTVLIMLSLLPMLEAQNPEHANITIGEPITNETLGWL...
  - ...
- Non-secreted proteins:
  - MAPPSVFAEVPQAQPVLVFKLIADFREDPDPRKVNLGVG...
  - MAHTLGLTQPNSTEPHKISFTAKEIDVIEWKDILVVG...
  - MSISESYAKEIKTAFRQFTDFPIEGEQFEDFLPIIGNP..
  - ...

Goal
- Build a classifier to predict whether new proteins are secreted or not.
String kernels

The idea

- Map each string $x \in \mathcal{X}$ to a vector $\Phi(x) \in \mathcal{F}$.
- Train a classifier for vectors on the images $\Phi(x_1), \ldots, \Phi(x_n)$ of the training set (nearest neighbor, linear perceptron, logistic regression, support vector machine...)

Diagram:

- $\mathcal{X}$ (strings)
- $\Phi$ (mapping to vectors)
- $\mathcal{F}$ (feature space)
- Examples of mapped strings: maskat..., msises, marssl..., malhtv..., mappsv..., mahtlg...
Example: substring indexation

The approach

Index the feature space by fixed-length strings, i.e.,

\[
\Phi (x) = (\Phi_u (x))_{u \in \mathcal{A}^k}
\]

where \( \Phi_u (x) \) can be:

- the number of occurrences of \( u \) in \( x \) (without gaps) : **spectrum kernel** (Leslie et al., 2002)
- the number of occurrences of \( u \) in \( x \) up to \( m \) mismatches (without gaps) : **mismatch kernel** (Leslie et al., 2004)
- the number of occurrences of \( u \) in \( x \) allowing gaps, with a weight decaying exponentially with the number of gaps : **substring kernel** (Lohdi et al., 2002)
The 3-spectrum of \( x = \text{CGGSLIAMMWFGV} \) is:

\[
(\text{CGG, GGS, GSL, SLI, LIA, IAM, AMM, MMW, MWF, WFG, FGV})
\]

Let \( \Phi_u(x) \) denote the number of occurrences of \( u \) in \( x \). The \( k \)-spectrum kernel is:

\[
K(x, x') := \sum_{u \in A^k} \Phi_u(x) \Phi_u(x')
\]
Implementation

The computation of the kernel is formally a sum over $|A|^k$ terms, but at most $|x| - k + 1$ terms are non-zero in $\Phi(x) \implies$ Computation in $O(|x| + |x'|)$ with pre-indexation of the strings.

Fast classification of a sequence $x$ in $O(|x|)$:

$$f(x) = w \cdot \Phi(x) = \sum_{u} w_u \Phi_u(x) = \sum_{i=1}^{\left| x \right| - k + 1} w_{x_i \ldots x_{i+k-1}}.$$

Remarks

- Work with any string (natural language, time series...)
- Fast and scalable, a good default method for string classification.
- Variants allow matching of $k$-mers up to $m$ mismatches.
Local alignment kernel (Saigo et al., 2004)

\[ s_{S,g}(\pi) = S(C, C) + S(L, L) + S(I, I) + S(A, V) + 2S(M, M) \]
\[ + S(W, W) + S(F, F) + S(G, G) + S(V, V) - g(3) - g(4) \]

\[ SW_{S,g}(x, y) := \max_{\pi \in \Pi(x, y)} s_{S,g}(\pi) \quad \text{is not a kernel} \]

\[ K_{LA}^{(\beta)}(x, y) = \sum_{\pi \in \Pi(x, y)} \exp(\beta s_{S,g}(x, y, \pi)) \quad \text{is a kernel} \]
Definition: Convolution kernel (Haussler, 1999)

Let $K_1$ and $K_2$ be two p.d. kernels for strings. The convolution of $K_1$ and $K_2$, denoted $K_1 \ast K_2$, is defined for any $x, x' \in X$ by:

$$K_1 \ast K_2(x, y) := \sum_{x_1 x_2 = x, y_1 y_2 = y} K_1(x_1, y_1) K_2(x_2, y_2).$$

Lemma

If $K_1$ and $K_2$ are p.d. then $K_1 \ast K_2$ is p.d.
\[ K_{LA}^{(\beta)} = \sum_{n=0}^{\infty} K_0 \ast \left( K_a^{(\beta)} \ast K_g^{(\beta)} \right)^{(n-1)} \ast K_a^{(\beta)} \ast K_0 , \]

with

- The constant kernel:
  \[ K_0 (x, y) := 1 . \]

- A kernel for letters:
  \[ K_a^{(\beta)} (x, y) := \begin{cases} 0 & \text{if } |x| \neq 1 \text{ where } |y| \neq 1 , \\ \exp (\beta S(x, y)) & \text{otherwise}. \end{cases} \]

- A kernel for gaps:
  \[ K_g^{(\beta)} (x, y) = \exp [\beta (g (|x|) + g (|x|))] . \]
The choice of kernel matters

Performance on the SCOP superfamily recognition benchmark (from Saigo et al., 2004).
Virtual screening for drug discovery

Image retrieval and classification

From Harchaoui and Bach (2007).
Graph kernels

1. Represent each graph $x$ by a vector $\Phi(x) \in \mathcal{H}$, either explicitly or implicitly through the kernel

$$K(x, x') = \Phi(x)^\top \Phi(x').$$

2. Use a linear method for classification in $\mathcal{H}$. 
1. Represent each graph $x$ by a vector $\Phi(x) \in \mathcal{H}$, either explicitly or implicitly through the kernel

$$K(x, x') = \Phi(x)^T \Phi(x').$$

2. Use a linear method for classification in $\mathcal{H}$. 
Graph kernels

1. Represent each graph $x$ by a vector $\Phi(x) \in \mathcal{H}$, either explicitly or implicitly through the kernel

$$K(x, x') = \Phi(x)^\top \Phi(x').$$

2. Use a linear method for classification in $\mathcal{H}$. 

Indexing by all subgraphs?

**Theorem**
Computing all subgraph occurrences is \textit{NP-hard}.

**Proof.**
- The linear graph of size \( n \) is a subgraph of a graph \( X \) with \( n \) vertices iff \( X \) has an Hamiltonian path.
- The decision problem whether a graph has a Hamiltonian path is \textit{NP-complete}. 
Computing all subgraph occurrences is NP-hard.

Proof.

- The linear graph of size $n$ is a subgraph of a graph $X$ with $n$ vertices iff $X$ has an Hamiltonian path.
- The decision problem whether a graph has a Hamiltonian path is NP-complete.
Indexing by all subgraphs?

Theorem
Computing all subgraph occurrences is $NP$-hard.

Proof.
- The linear graph of size $n$ is a subgraph of a graph $X$ with $n$ vertices iff $X$ has an Hamiltonian path.
- The decision problem whether a graph has a Hamiltonian path is $NP$-complete.
Indexing by specific subgraphs

Substructure selection

We can imagine more limited sets of substructures that lead to more computationally efficient indexing (non-exhaustive list)

- substructures selected by domain knowledge (MDL fingerprint)
- all path up to length $k$ (Openeye fingerprint, Nicholls 2005)
- all shortest paths (Borgwardt and Kriegel, 2005)
- all subgraphs up to $k$ vertices (graphlet kernel, Sherashidze et al., 2009)
- all frequent subgraphs in the database (Helma et al., 2004)
Example: Indexing by all shortest paths

Properties (Borgwardt and Kriegel, 2005)

- There are $O(n^2)$ shortest paths.
- The vector of counts can be computed in $O(n^4)$ with the Floyd-Warshall algorithm.
Example: Indexing by all shortest paths

There are $O(n^2)$ shortest paths.

The vector of counts can be computed in $O(n^4)$ with the Floyd-Warshall algorithm.
Example: Indexing by all subgraphs up to $k$ vertices

Properties (Shervashidze et al., 2009)

- Naive enumeration scales as $O(n^k)$.  
- Enumeration of connected graphlets in $O(nd^{k-1})$ for graphs with degree $\leq d$ and $k \leq 5$.  
- Randomly sample subgraphs if enumeration is infeasible.
Example: Indexing by all subgraphs up to $k$ vertices

Properties (Shervashidze et al., 2009)

- Naive enumeration scales as $O(n^k)$.
- Enumeration of connected graphlets in $O(nd^{k-1})$ for graphs with degree $\leq d$ and $k \leq 5$.
- Randomly sample subgraphs if enumeration is infeasible.
A walk of a graph \((V, E)\) is sequence of \(v_1, \ldots, v_n \in V\) such that 
\((v_i, v_{i+1}) \in E\) for \(i = 1, \ldots, n - 1\).

We note \(W_n(G)\) the set of walks with \(n\) vertices of the graph \(G\), and \(W(G)\) the set of all walks.
Walks $\neq$ paths
Walk kernel

**Definition**

- Let $S_n$ denote the set of all possible label sequences of walks of length $n$ (including vertices and edges labels), and $S = \bigcup_{n \geq 1} S_n$.
- For any graph $X$ let a weight $\lambda_G(w)$ be associated to each walk $w \in \mathcal{W}(G)$.
- Let the feature vector $\Phi(G) = (\Phi_s(G))_{s \in S}$ be defined by:

\[
\Phi_s(G) = \sum_{w \in \mathcal{W}(G)} \lambda_G(w) 1 \ (s \text{ is the label sequence of } w).
\]

- A walk kernel is a graph kernel defined by:

\[
K_{\text{walk}}(G_1, G_2) = \sum_{s \in S} \Phi_s(G_1) \Phi_s(G_2).
\]
Walk kernel

Definition

Let $S_n$ denote the set of all possible label sequences of walks of length $n$ (including vertices and edges labels), and $S = \bigcup_{n \geq 1} S_n$.

For any graph $\mathcal{G}$ let a weight $\lambda_G(w)$ be associated to each walk $w \in \mathcal{W}(G)$.

Let the feature vector $\Phi(G) = (\Phi_s(G))_{s \in S}$ be defined by:

$$\Phi_s(G) = \sum_{w \in \mathcal{W}(G)} \lambda_G(w)1 \ (s \text{ is the label sequence of } w) .$$

A walk kernel is a graph kernel defined by:

$$K_{\text{walk}}(G_1, G_2) = \sum_{s \in S} \Phi_s(G_1)\Phi_s(G_2) .$$
Walk kernel examples

- The *nth-order walk kernel* is the walk kernel with \( \lambda_G(w) = 1 \) if the length of \( w \) is \( n \), 0 otherwise. It compares two graphs through their common walks of length \( n \).

- The *random walk kernel* is obtained with \( \lambda_G(w) = P_G(w) \), where \( P_G \) is a Markov random walk on \( G \). In that case we have:

\[
K(G_1, G_2) = P(label(W_1) = label(W_2)),
\]

where \( W_1 \) and \( W_2 \) are two independant random walks on \( G_1 \) and \( G_2 \), respectively (Kashima et al., 2003).

- The *geometric walk kernel* is obtained (when it converges) with \( \lambda_G(w) = \beta^{length(w)} \), for \( \beta > 0 \). In that case the feature space is of infinite dimension (Gärtner et al., 2003).
Walk kernel examples

- The $n$th-order walk kernel is the walk kernel with $\lambda_G(w) = 1$ if the length of $w$ is $n$, 0 otherwise. It compares two graphs through their common walks of length $n$.

- The random walk kernel is obtained with $\lambda_G(w) = P_G(w)$, where $P_G$ is a Markov random walk on $G$. In that case we have:

$$K(G_1, G_2) = P(label(W_1) = label(W_2)),$$

where $W_1$ and $W_2$ are two independent random walks on $G_1$ and $G_2$, respectively (Kashima et al., 2003).

- The geometric walk kernel is obtained (when it converges) with $\lambda_G(w) = \beta^{\text{length}(w)}$, for $\beta > 0$. In that case the feature space is of infinite dimension (Gärtner et al., 2003).
Walk kernel examples

- The $n$th-order walk kernel is the walk kernel with $\lambda_G(w) = 1$ if the length of $w$ is $n$, 0 otherwise. It compares two graphs through their common walks of length $n$.

- The random walk kernel is obtained with $\lambda_G(w) = P_G(w)$, where $P_G$ is a Markov random walk on $G$. In that case we have:

$$K(G_1, G_2) = P(label(W_1) = label(W_2)),$$

where $W_1$ and $W_2$ are two independent random walks on $G_1$ and $G_2$, respectively (Kashima et al., 2003).

- The geometric walk kernel is obtained (when it converges) with $\lambda_G(w) = \beta^{\text{length}(w)}$, for $\beta > 0$. In that case the feature space is of infinite dimension (Gärtner et al., 2003).
Computation of walk kernels

**Proposition**

These three kernels (\(n\)th-order, random and geometric walk kernels) can be computed efficiently in \textit{polynomial time}.
Product graph

**Definition**

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs with labeled vertices. The **product graph** $G = G_1 \times G_2$ is the graph $G = (V, E)$ with:

1. $V = \{(v_1, v_2) \in V_1 \times V_2 : v_1 \text{ and } v_2 \text{ have the same label}\}$,
2. $E = \{((v_1, v_2), (v'_1, v'_2)) \in V \times V : (v_1, v'_1) \in E_1 \text{ and } (v_2, v'_2) \in E_2\}$.

---

G1

- 1
- 2
- 3
- 4

G2

- a
- b
- c
- d
- e

G1 $\times$ G2

- 1b 2a 1d
- 3c 2d
- 3e 4c 4e
Walk kernel and product graph

**Lemma**

There is a bijection between:

1. The pairs of walks $w_1 \in \mathcal{W}_n(G_1)$ and $w_2 \in \mathcal{W}_n(G_2)$ with the same label sequences,
2. The walks on the product graph $w \in \mathcal{W}_n(G_1 \times G_2)$.

**Corollary**

$$K_{walk}(G_1, G_2) = \sum_{s \in S} \Phi_s(G_1)\Phi_s(G_2) = \sum_{(w_1, w_2) \in \mathcal{W}(G_1) \times \mathcal{W}(G_1)} \lambda_{G_1}(w_1)\lambda_{G_2}(w_2)1(l(w_1) = l(w_2)) = \sum_{w \in \mathcal{W}(G_1 \times G_2)} \lambda_{G_1 \times G_2}(w).$$
Walk kernel and product graph

Lemma

There is a bijection between:

1. The pairs of walks $w_1 \in \mathcal{W}_n(G_1)$ and $w_2 \in \mathcal{W}_n(G_2)$ with the same label sequences,
2. The walks on the product graph $w \in \mathcal{W}_n(G_1 \times G_2)$.

Corollary

\[
K_{walk}(G_1, G_2) = \sum_{s \in S} \Phi_s(G_1)\Phi_s(G_2)
\]

\[
= \sum_{(w_1, w_2) \in \mathcal{W}(G_1) \times \mathcal{W}(G_1)} \lambda_{G_1}(w_1)\lambda_{G_2}(w_2)1(l(w_1) = l(w_2))
\]

\[
= \sum_{w \in \mathcal{W}(G_1 \times G_2)} \lambda_{G_1 \times G_2}(w).
\]
Computation of the $n$th-order walk kernel

For the $n$th-order walk kernel we have $\lambda_{G_1 \times G_2}(w) = 1$ if the length of $w$ is $n$, 0 otherwise.

Therefore:

$$K_{nth-order}(G_1, G_2) = \sum_{w \in \mathcal{W}_n(G_1 \times G_2)} 1.$$ 

Let $A$ be the adjacency matrix of $G_1 \times G_2$. Then we get:

$$K_{nth-order}(G_1, G_2) = \sum_{i,j} [A^n]_{i,j} = 1^\top A^n 1.$$ 

Computation in $O(n|G_1||G_2|d_1d_2)$, where $d_i$ is the maximum degree of $G_i$. 
Computation of random and geometric walk kernels

- In both cases $\lambda_G(w)$ for a walk $w = v_1 \ldots v_n$ can be decomposed as:

  $$
  \lambda_G(v_1 \ldots v_n) = \lambda^i(v_1) \prod_{i=2}^{n} \lambda^t(v_{i-1}, v_i).
  $$

- Let $\Lambda_i$ be the vector of $\lambda^i(v)$ and $\Lambda_t$ be the matrix of $\lambda^t(v, v')$:

  $$
  K_{\text{walk}}(G_1, G_2) = \sum_{n=1}^{\infty} \sum_{w \in \mathcal{W}_n(G_1 \times G_2)} \lambda^i(v_1) \prod_{i=2}^{n} \lambda^t(v_{i-1}, v_i)
  $$

  $$
  = \sum_{n=0}^{\infty} \Lambda_i \Lambda_t^n \mathbf{1}
  $$

  $$
  = \Lambda_i (I - \Lambda_t)^{-1} \mathbf{1}
  $$

- Computation in $O(|G_1|^3 |G_2|^3)$
Extension: branching walks (Ramon and Gärtner, 2003; Mahé and Vert, 2009)

\[
\mathcal{T}(v, n + 1) = \sum_{R \subseteq \mathcal{N}(v)} \prod_{v' \in R} \lambda_t(v, v') \mathcal{T}(v', n),
\]
Screening of inhibitors for 60 cancer cell lines.
Image classification (Harchaoui and Bach, 2007)

COREL14 dataset

- 1400 natural images in 14 classes
- Compare kernel between histograms (H), walk kernel (W), subtree kernel (TW), weighted subtree kernel (wTW), and a combination (M).

Performance comparison on Corel14


A. Nicholls. Oechem, version 1.3.4, openeye scientific software. website, 2005.

