# S1324:
# Support Vector Machines

**Assoc. Prof. Sascha Hornauer, based on slides from**

**Chloé-Agathe Azencott**

chloe-agathe.azencott@mines-paristech.fr

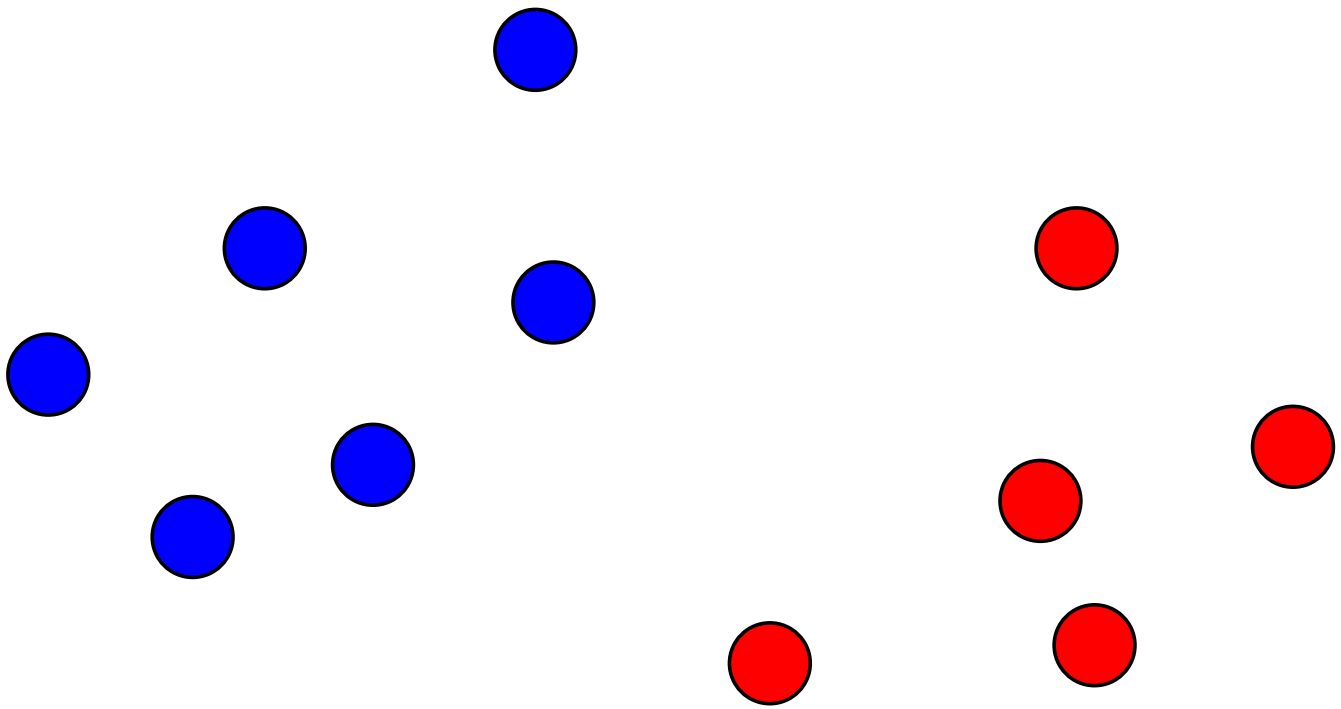
based on slides by J.-P. Vert

2015-05-22

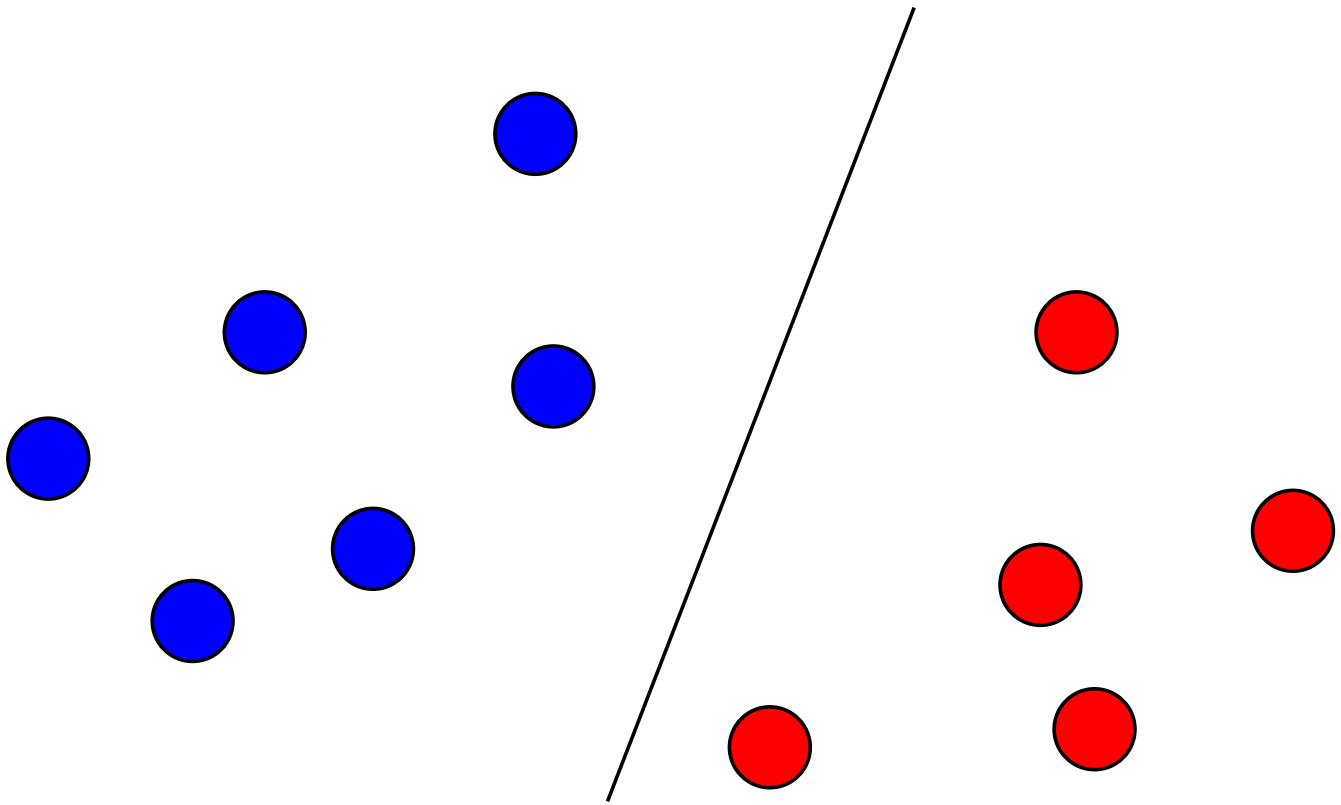


# Hard-margin Linear Support Vector Machines

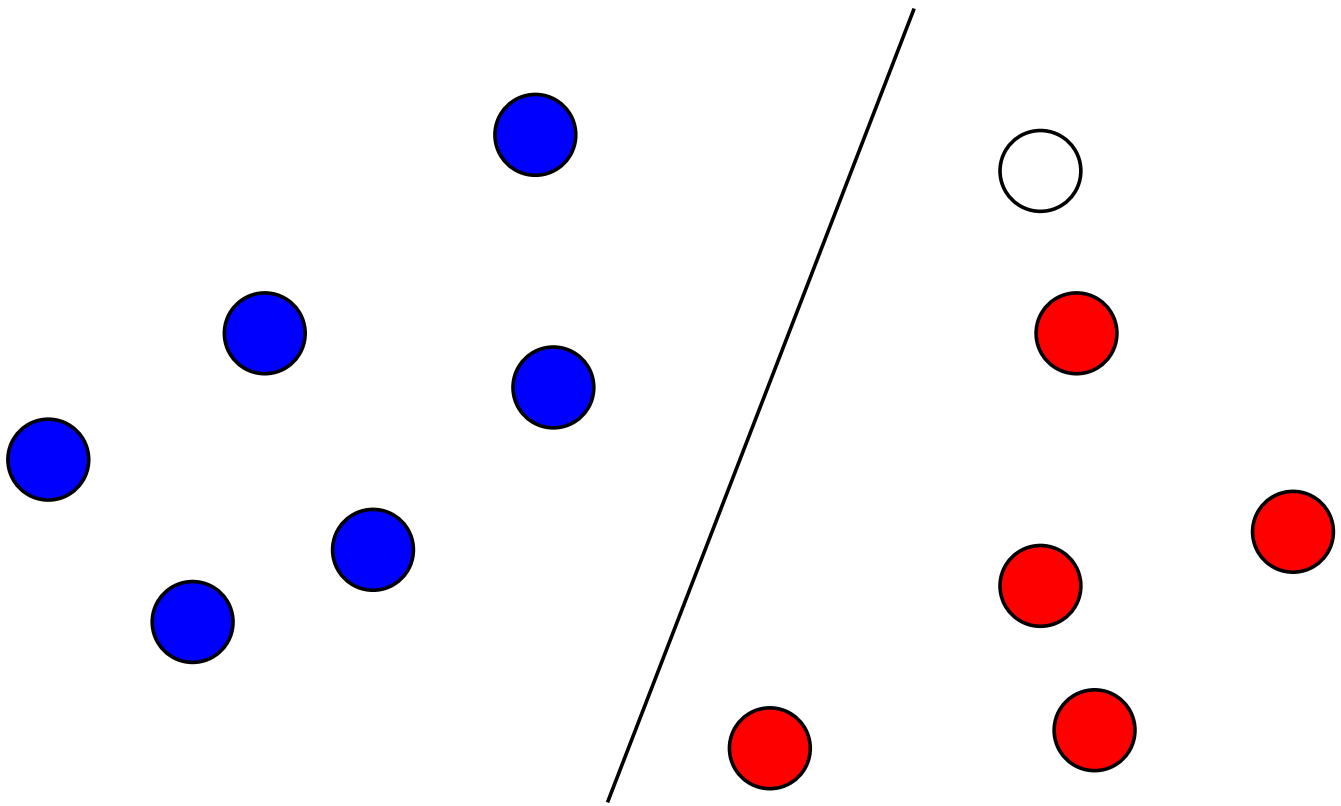
# Linear classifier



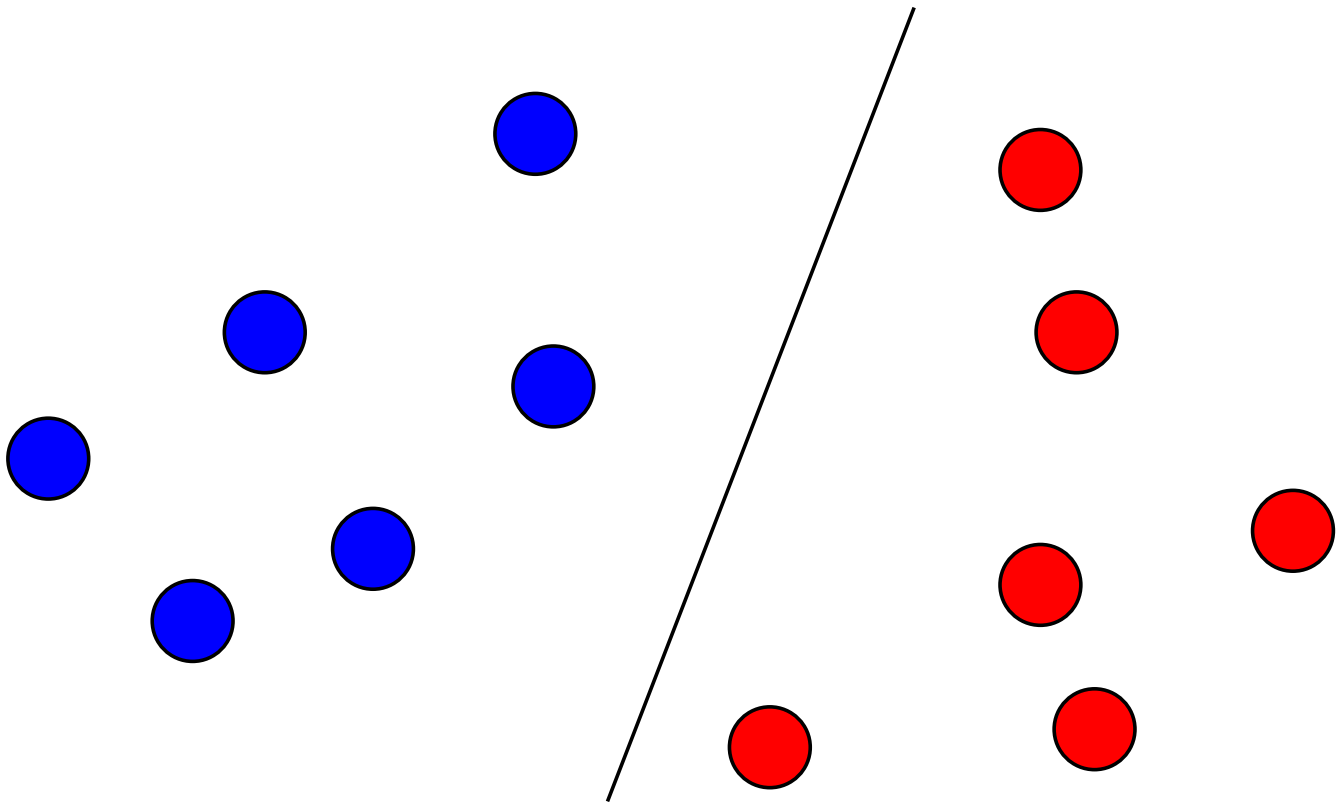
# Linear classifier



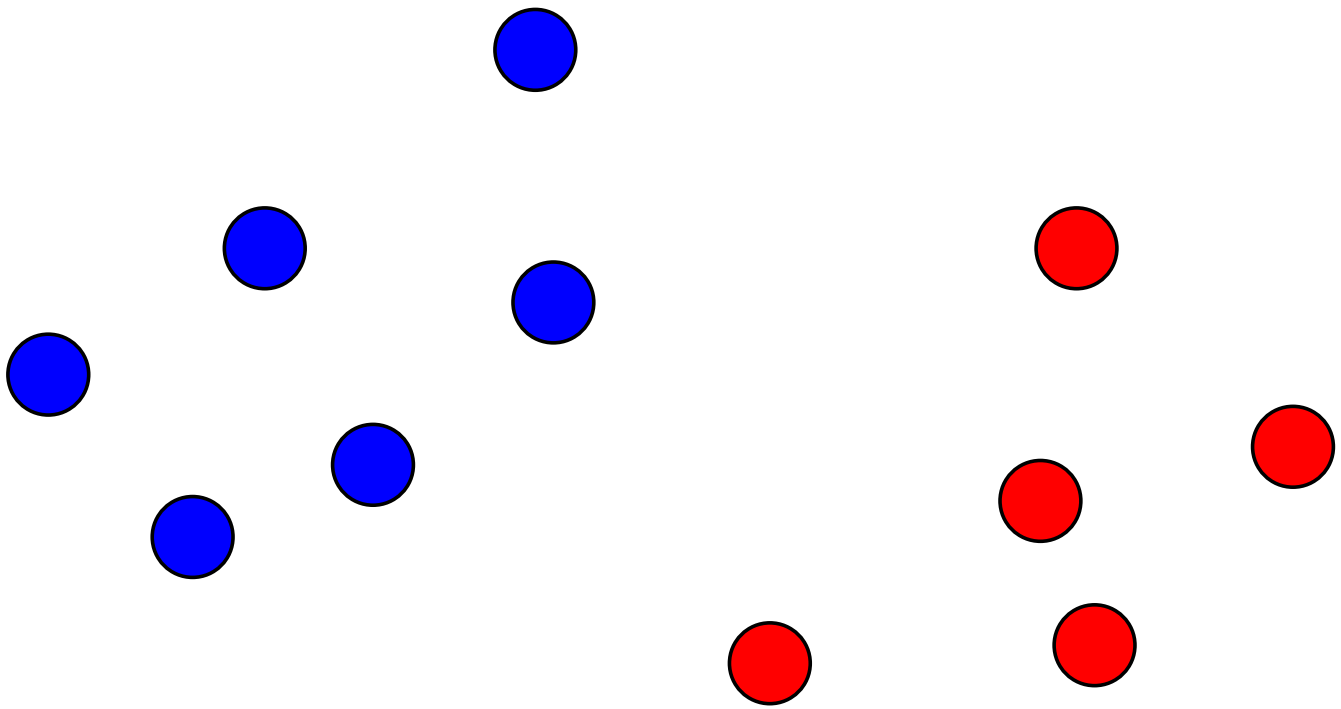
# Linear classifier



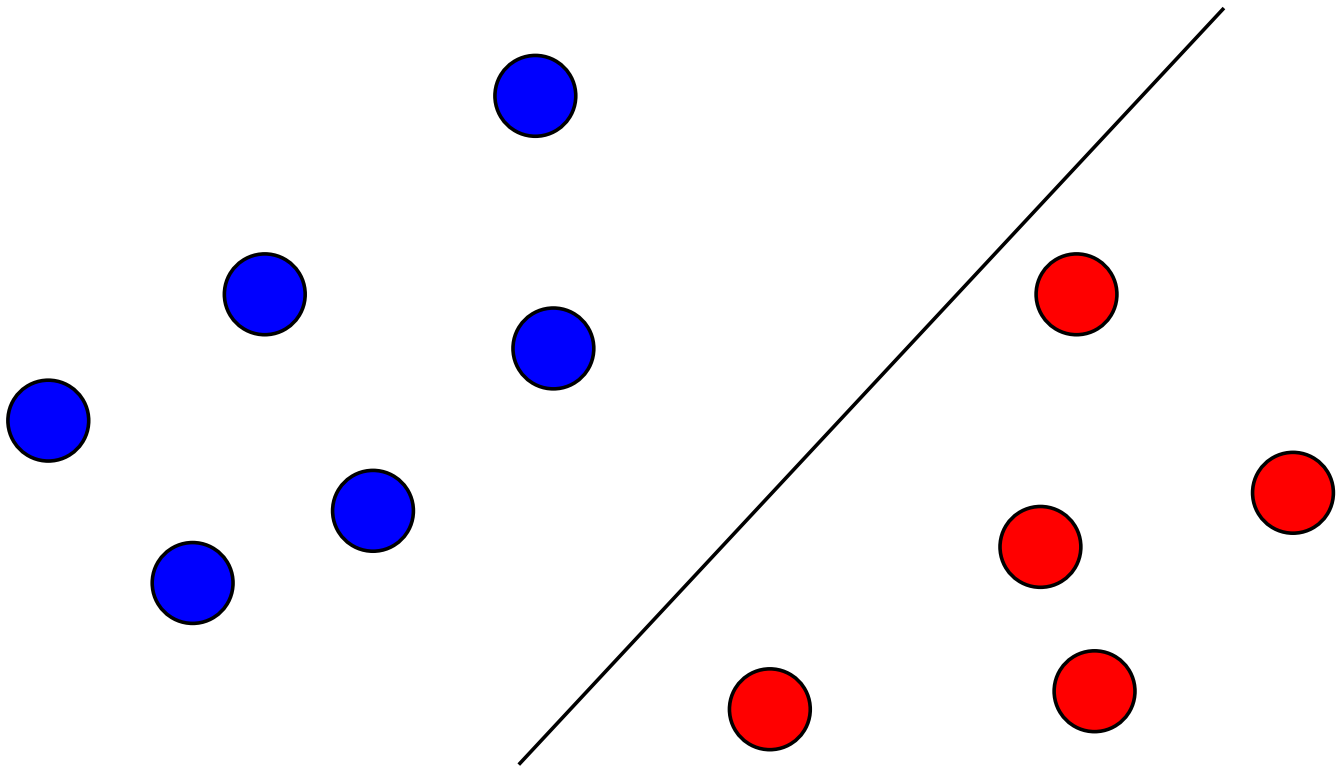
# Linear classifier



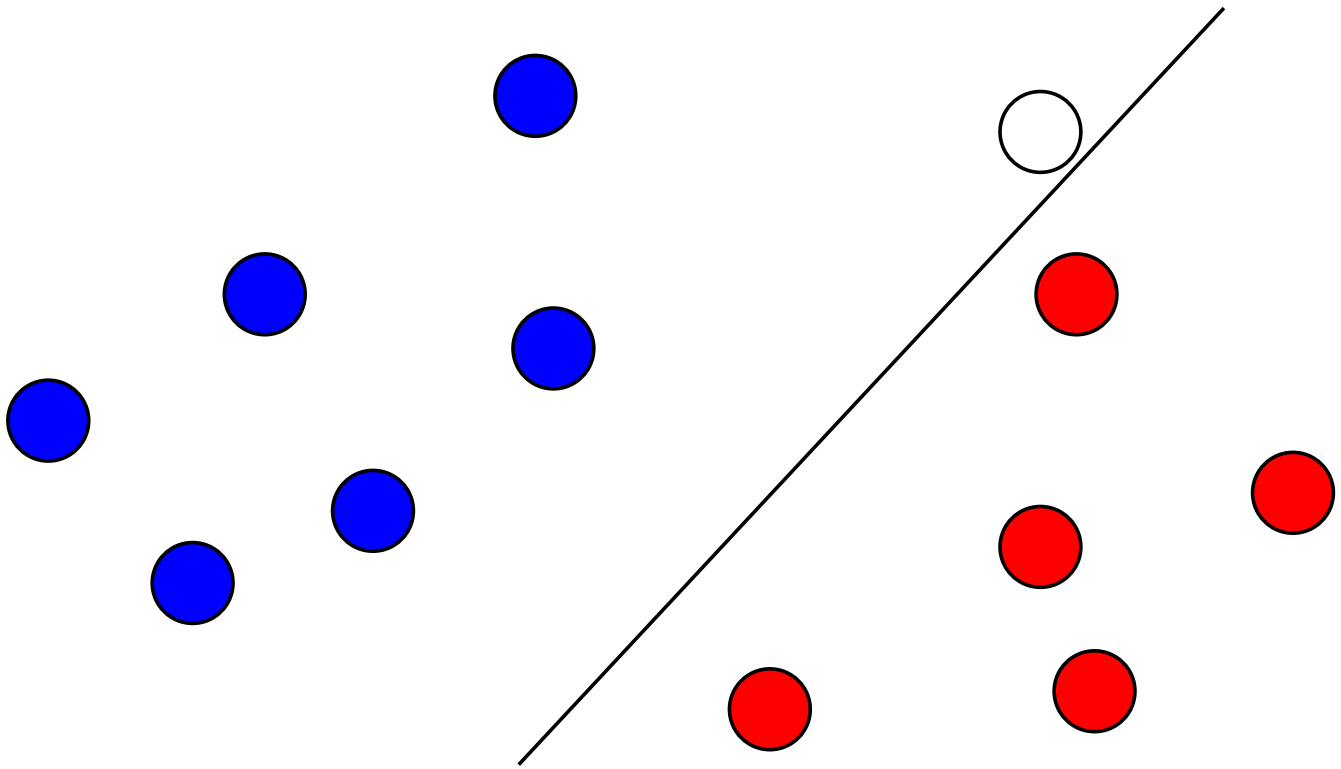
# Linear classifier



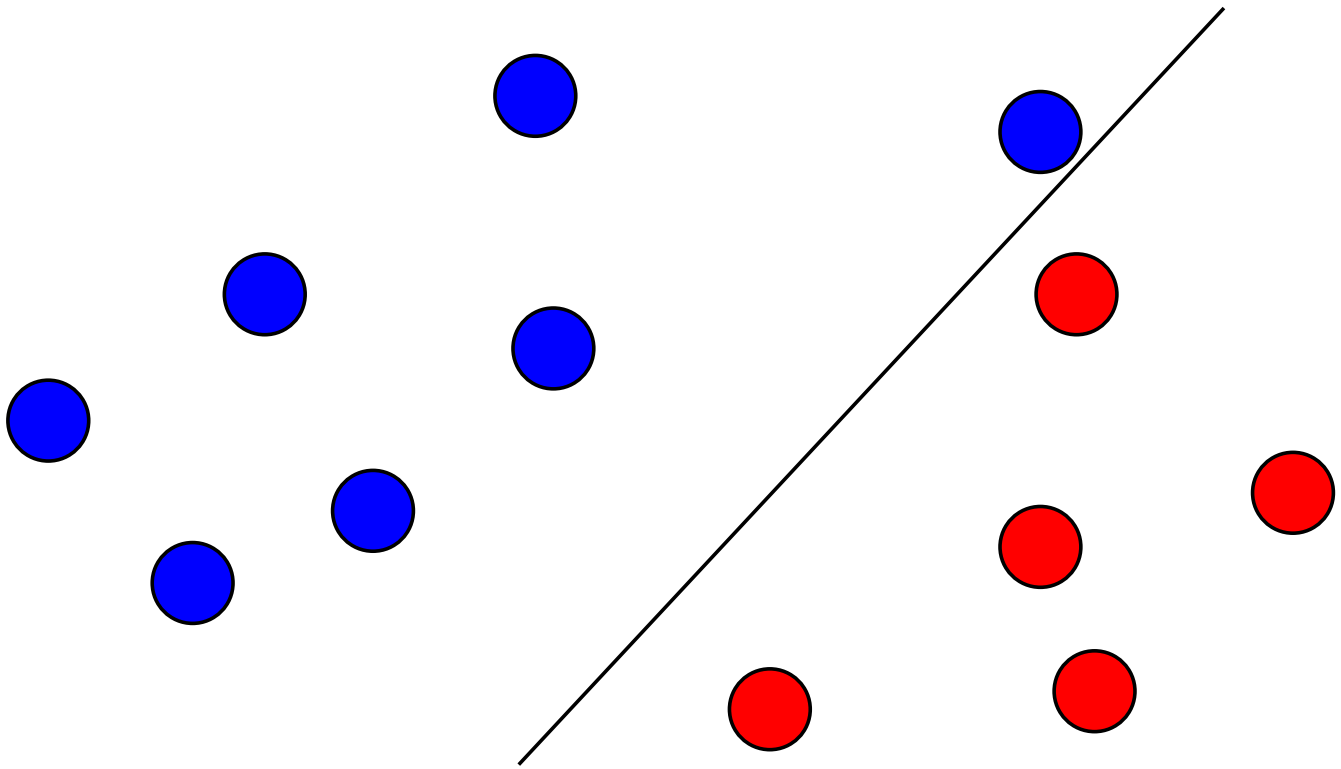
# Linear classifier



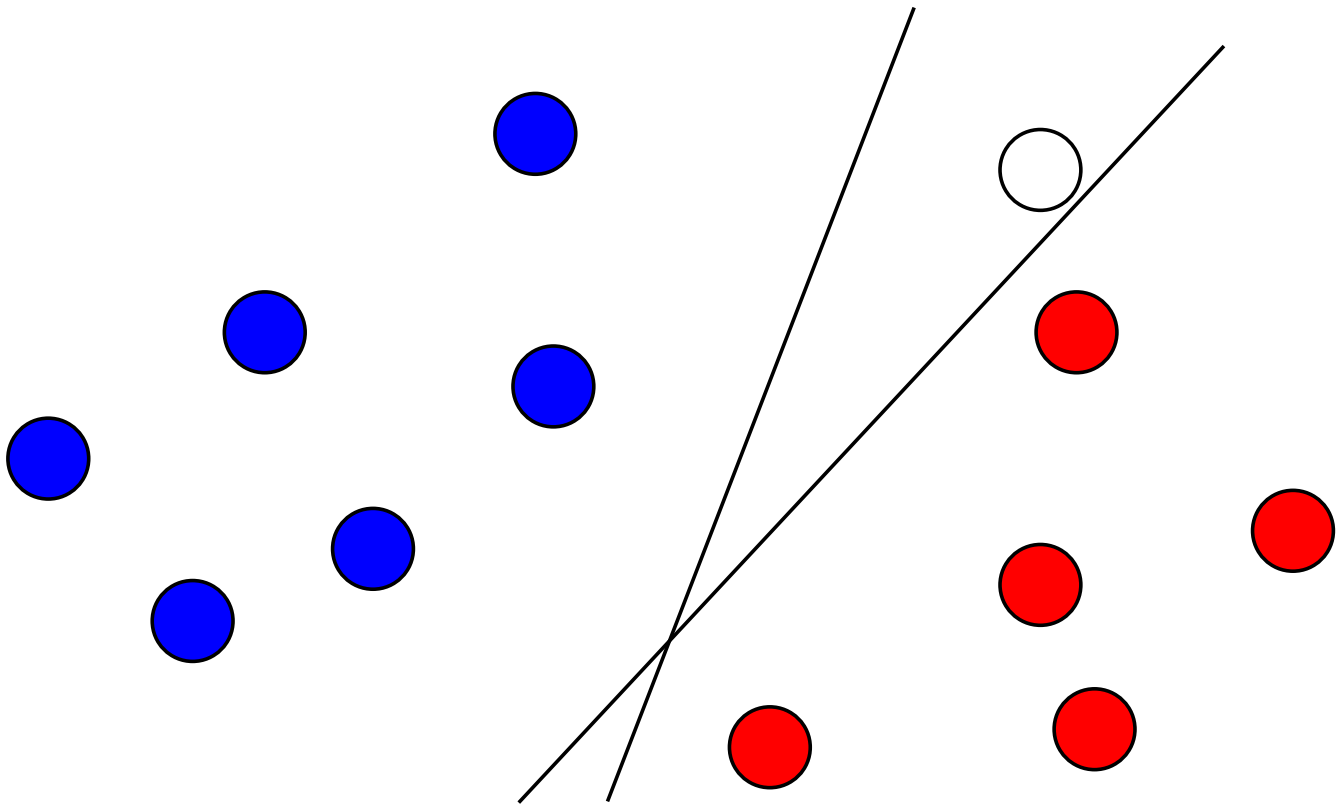
# Linear classifier



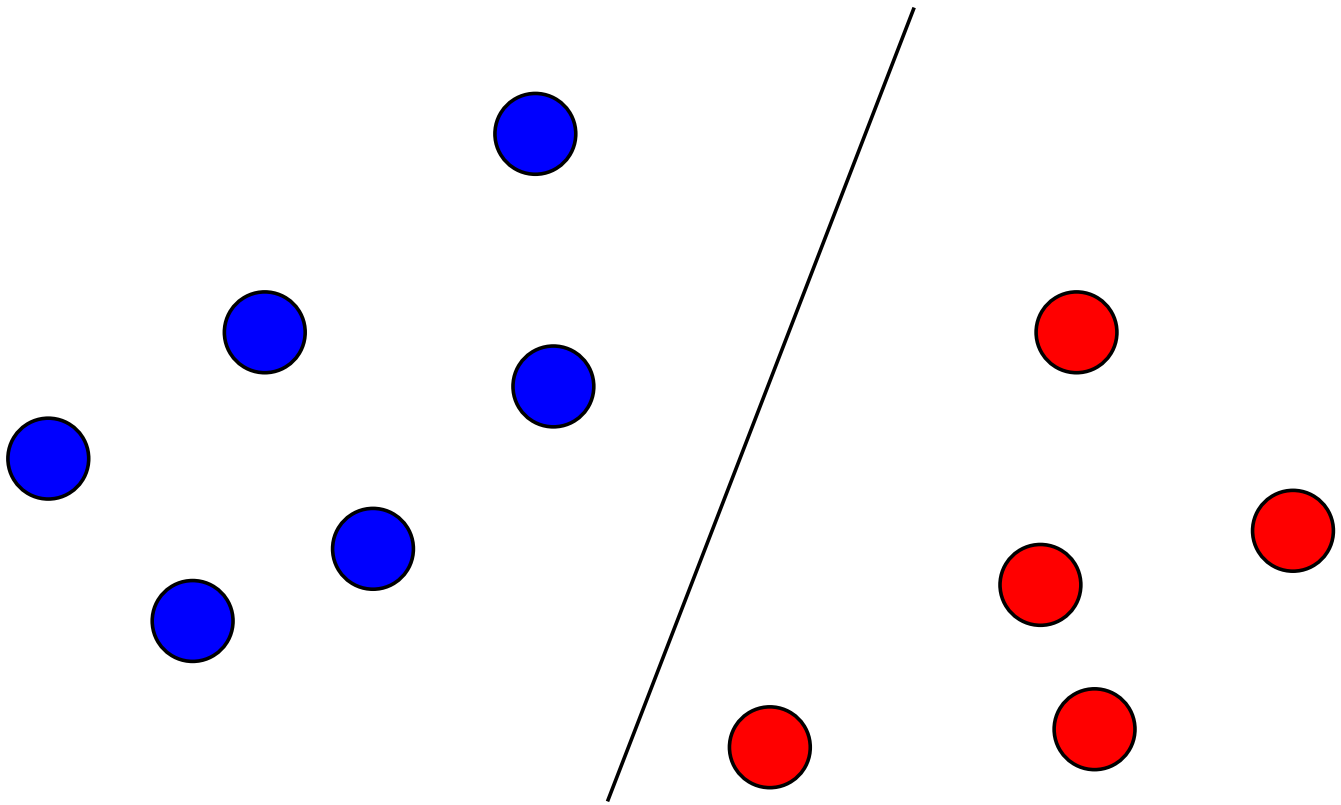
# Linear classifier



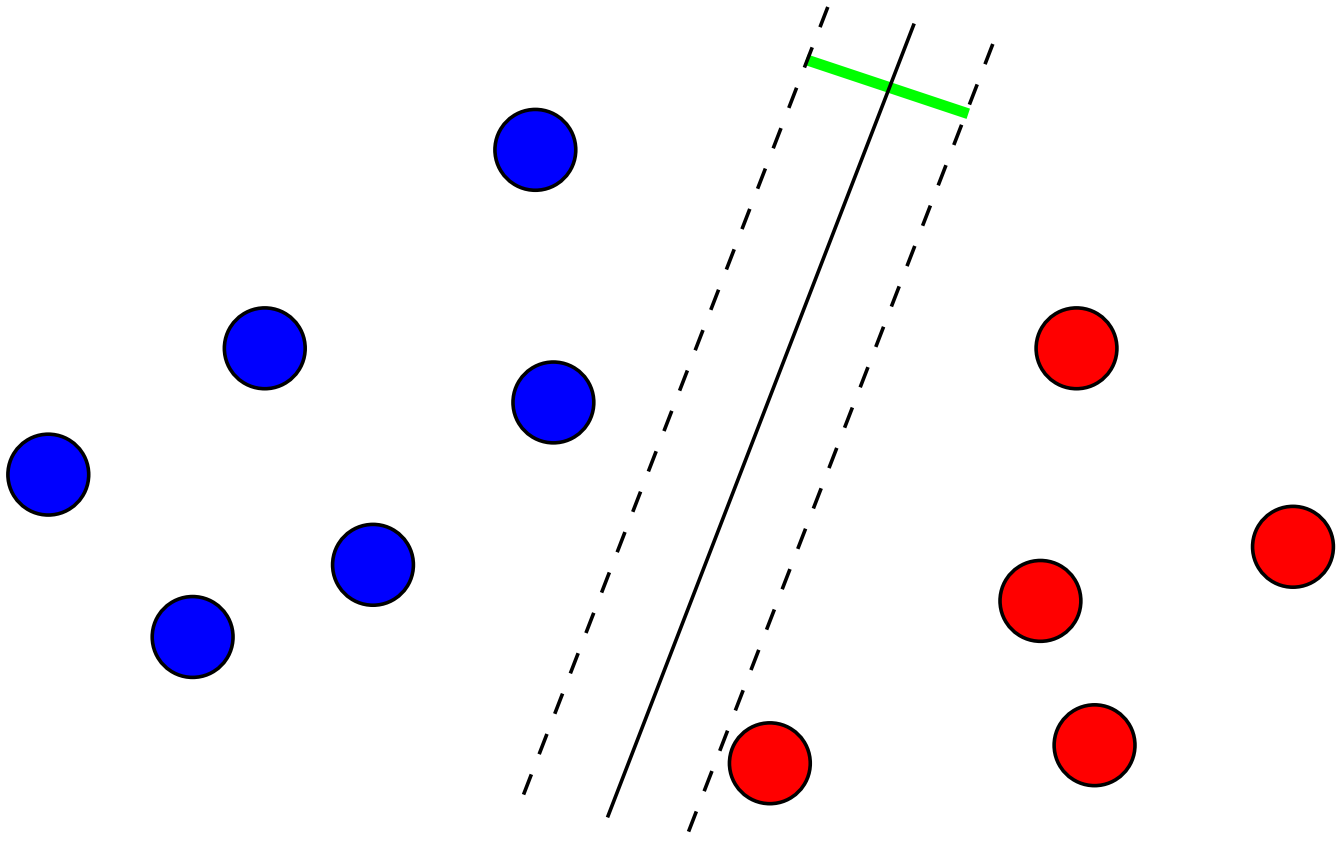
# Which one is better?



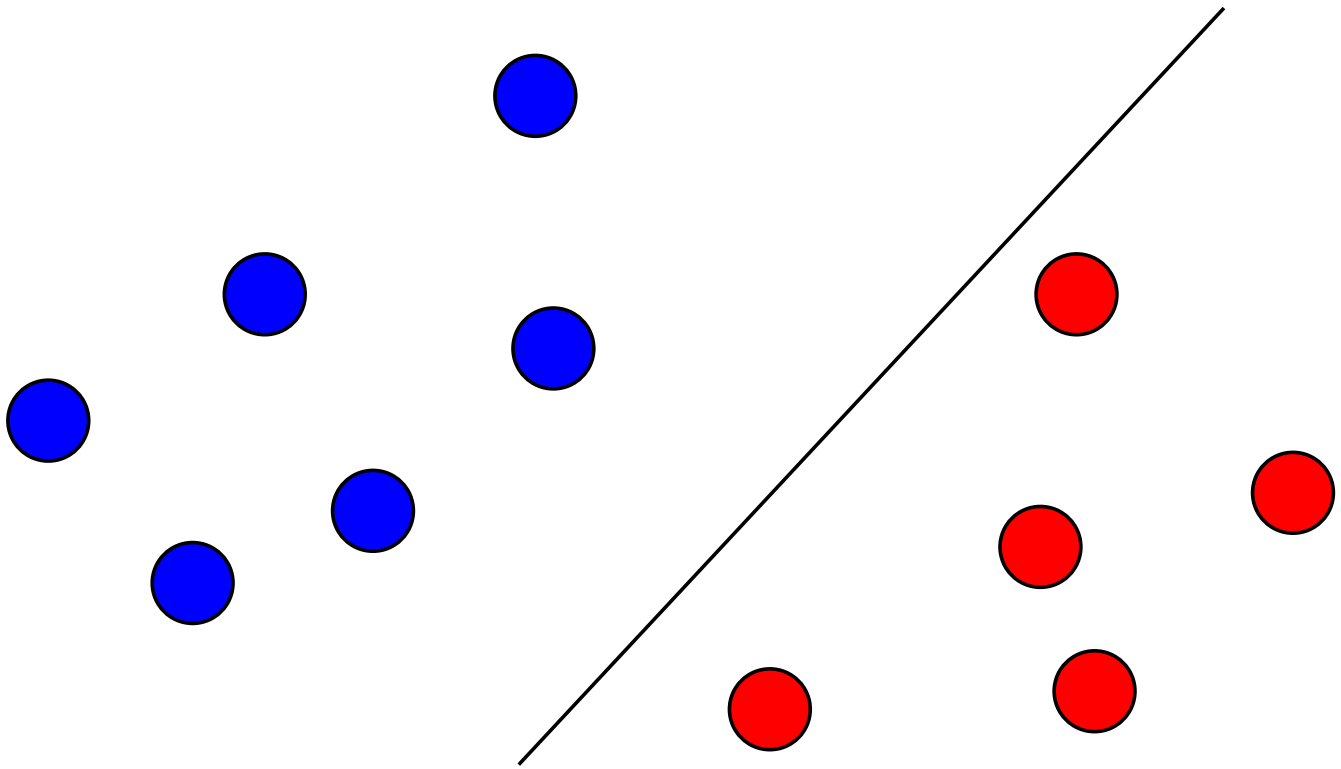
# The margin of a linear classifier



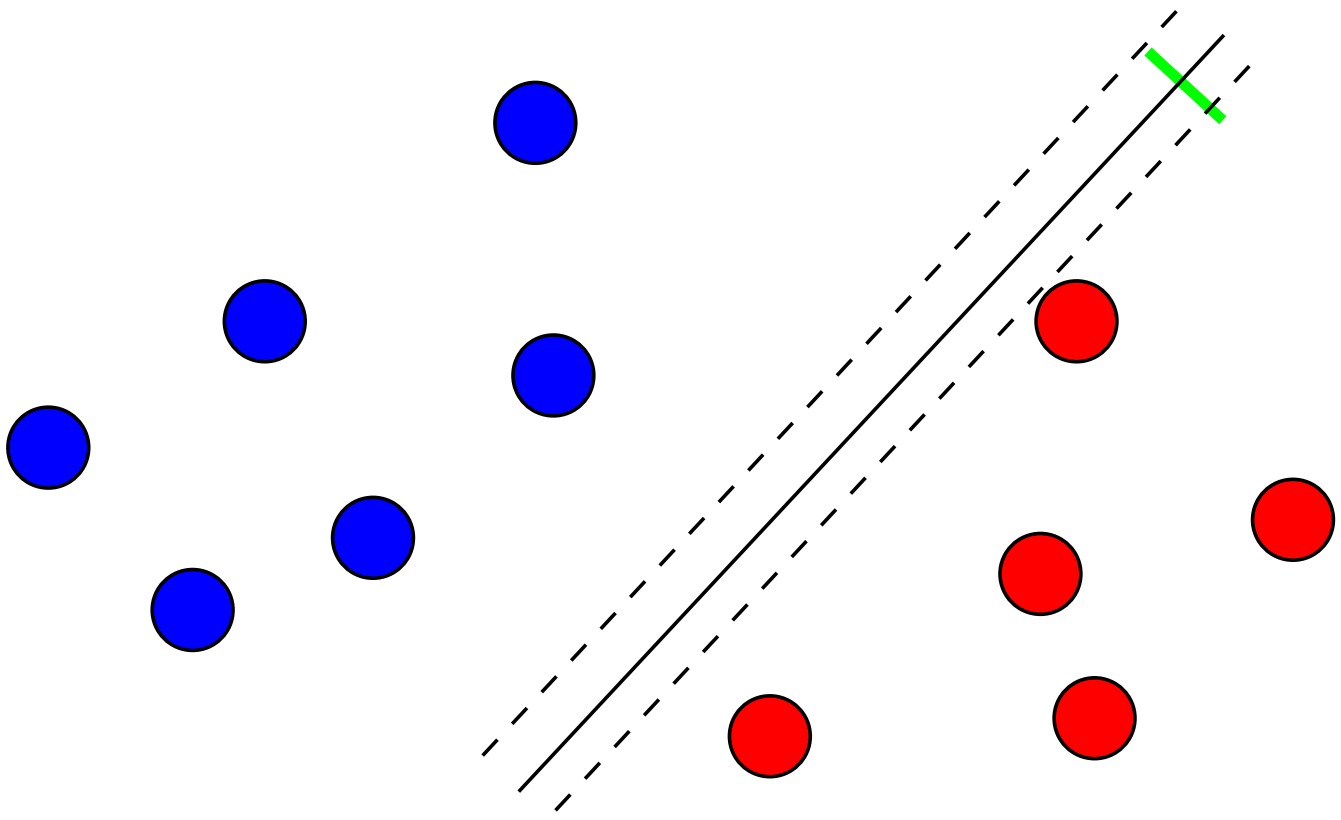
# The margin of a linear classifier



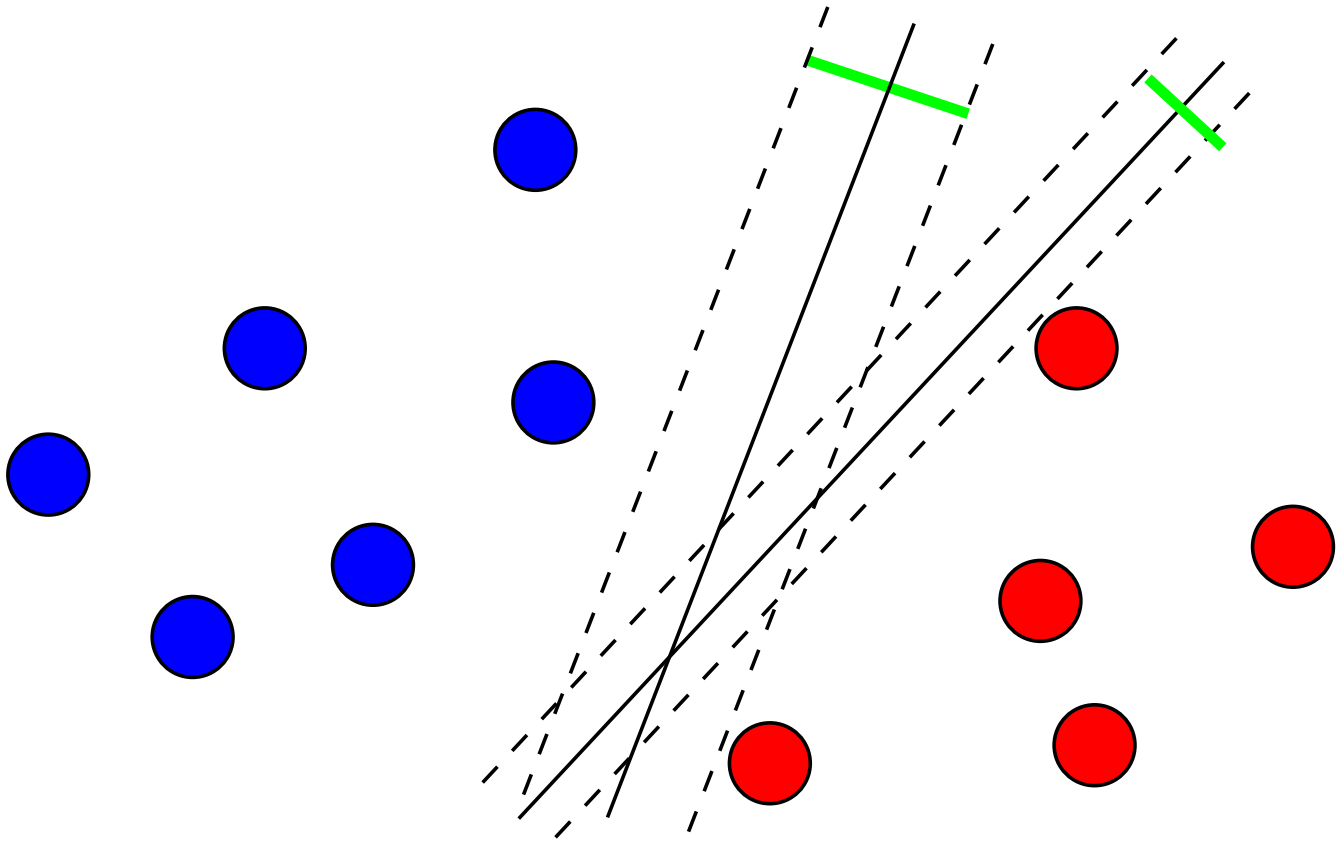
# The margin of a linear classifier



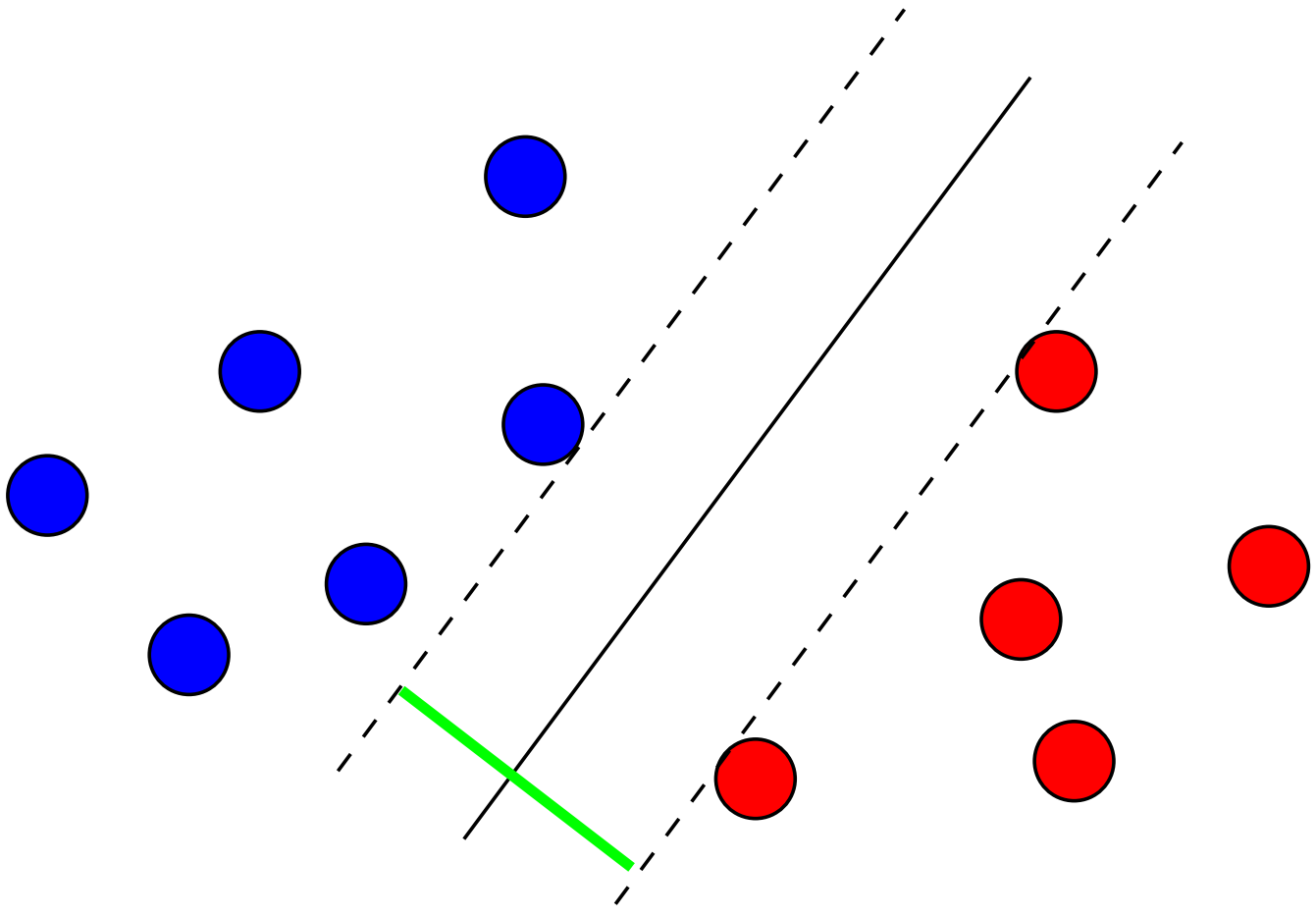
# The margin of a linear classifier



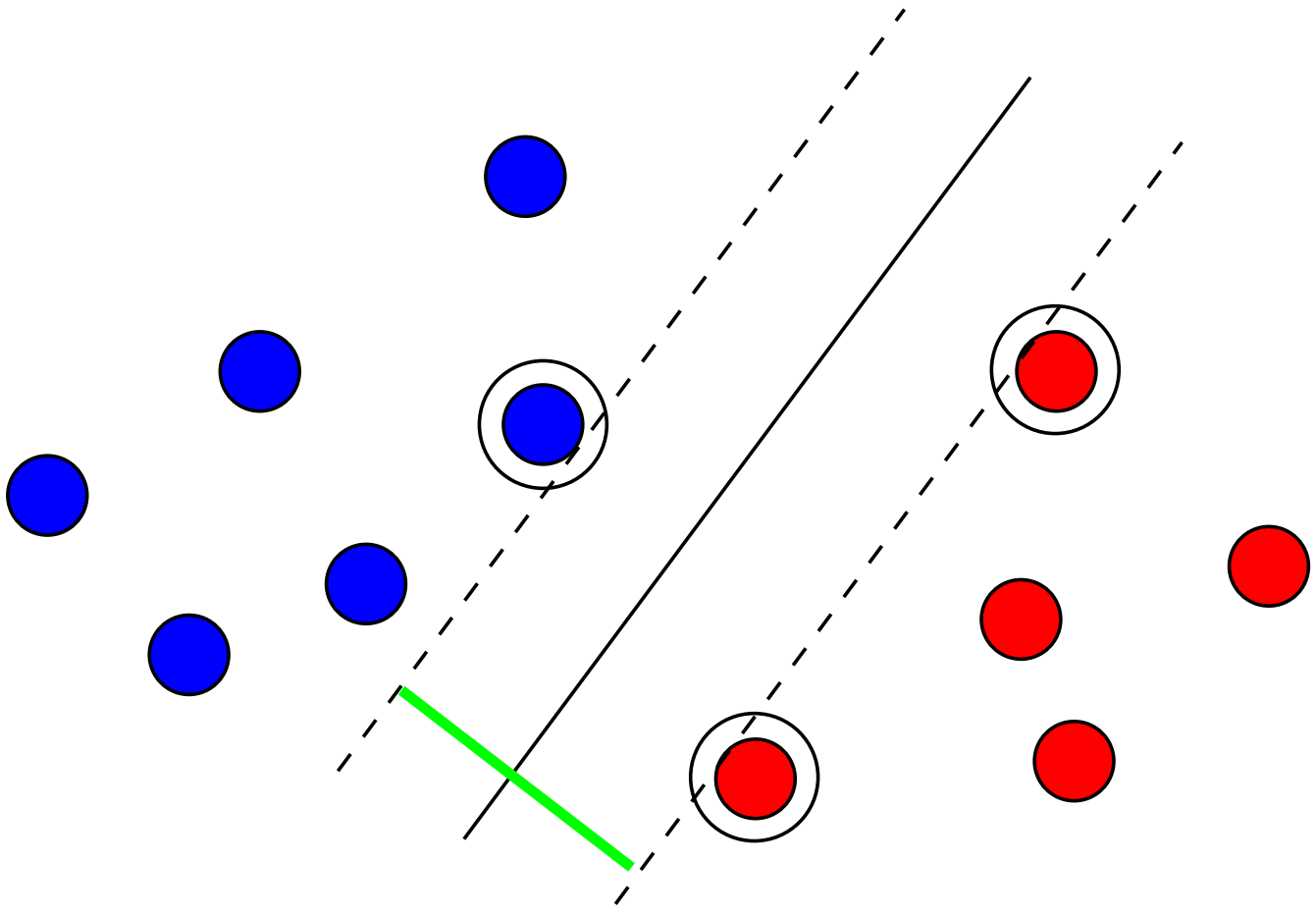
# The margin of a linear classifier



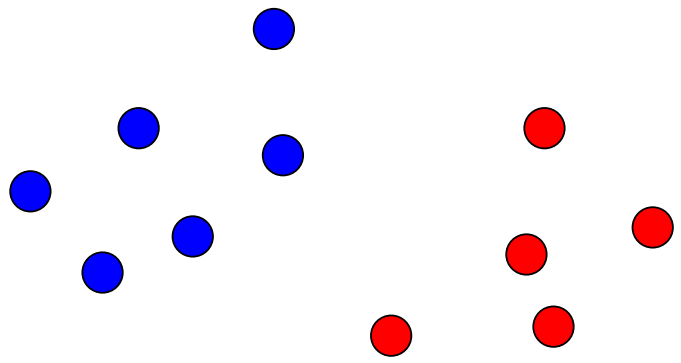
# Largest margin classifier (*hard-margin SVM*)



# Support vectors



# More formally



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

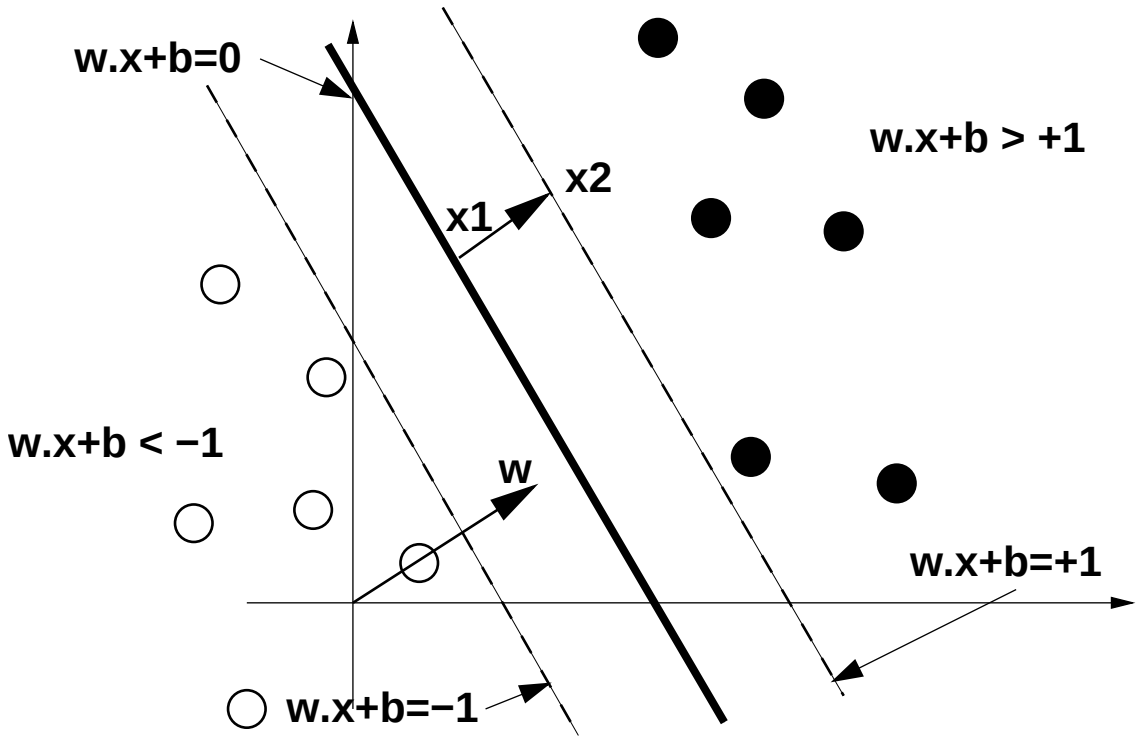
$$\mathcal{S} = \left\{(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)\right\} ,$$

  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:

$$\begin{cases} \vec{w}.\vec{x}_i + b > 0 & \text{if } y_i = 1 , \\ \vec{w}.\vec{x}_i + b < 0 & \text{if } y_i = -1 . \end{cases}$$

# How to find the largest separating hyperplane?

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



# The margin is $2/\| \vec{w} \|$

Indeed, the points $\vec{x}_1$ and $\vec{x}_2$ satisfy:

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

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:

$$\vec{w}.\vec{x}_i + b \geq 1\,.$$

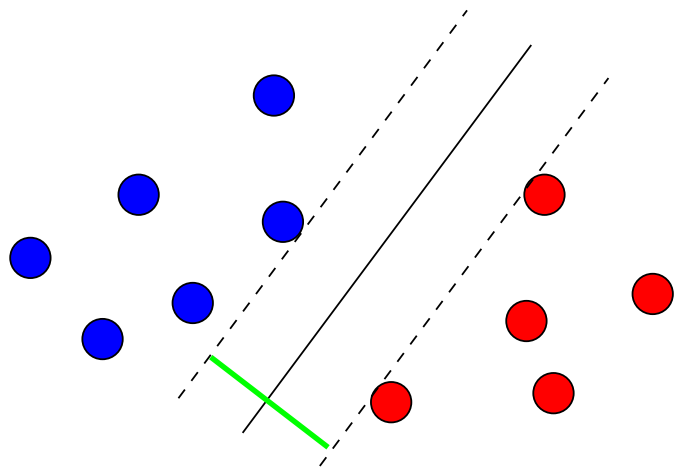
For negative examples ($y_i = -1$) this means:

$$\vec{w}.\vec{x}_i + b \leq -1\,.$$

Both cases are summarized by:

$$\forall i = 1, \ldots, n\,, \qquad y_i\left(\vec{w}.\vec{x}_i + b\right) \geq 1\,.$$

# Finding the optimal hyperplane



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

$$\| \vec{w} \|^2$$

under the constraints:

$$\forall i = 1, \ldots, n, \qquad y_i \left( \vec{w}.\vec{x}_i + b \right) - 1 \geq 0 \, .$$

*This is a classical quadratic program on* $\mathbb{R}^{p+1}$.

# Lagrangian

In order to minimize:

$$\frac{1}{2}\| \vec{w} \|_2^2$$

under the constraints:

$$\forall i = 1, \ldots, n\,, \qquad y_i \left(\vec{w}.\vec{x}_i + b\right) - 1 \geq 0\,,$$

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

$$L\left(\vec{w}, b, \vec{\alpha}\right) = \frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{n} \alpha_i \left(y_i \left(\vec{w}.\vec{x}_i + b\right) - 1\right)\,.$$

# Lagrangian

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

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

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

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

# Dual function

- We therefore obtain the Lagrange dual function:

$$q(\vec{\alpha}) = \inf_{\vec{w}\in\mathbb{R}^p, b\in\mathbb{R}} L(\vec{w}, b, \vec{\alpha})$$
$$= \begin{cases} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \vec{x}_i . \vec{x}_j & \text{if } \sum_{i=1}^{n} \alpha_i y_i = 0\,, \\ -\infty & \text{otherwise.} \end{cases}$$

- The dual problem is:

$$\begin{aligned} \text{maximize} \quad & q(\vec{\alpha}) \\ \text{subject to} \quad & \vec{\alpha} \geq 0\,. \end{aligned}$$

# Dual problem

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 (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.*

# Recovering the optimal hyperplane

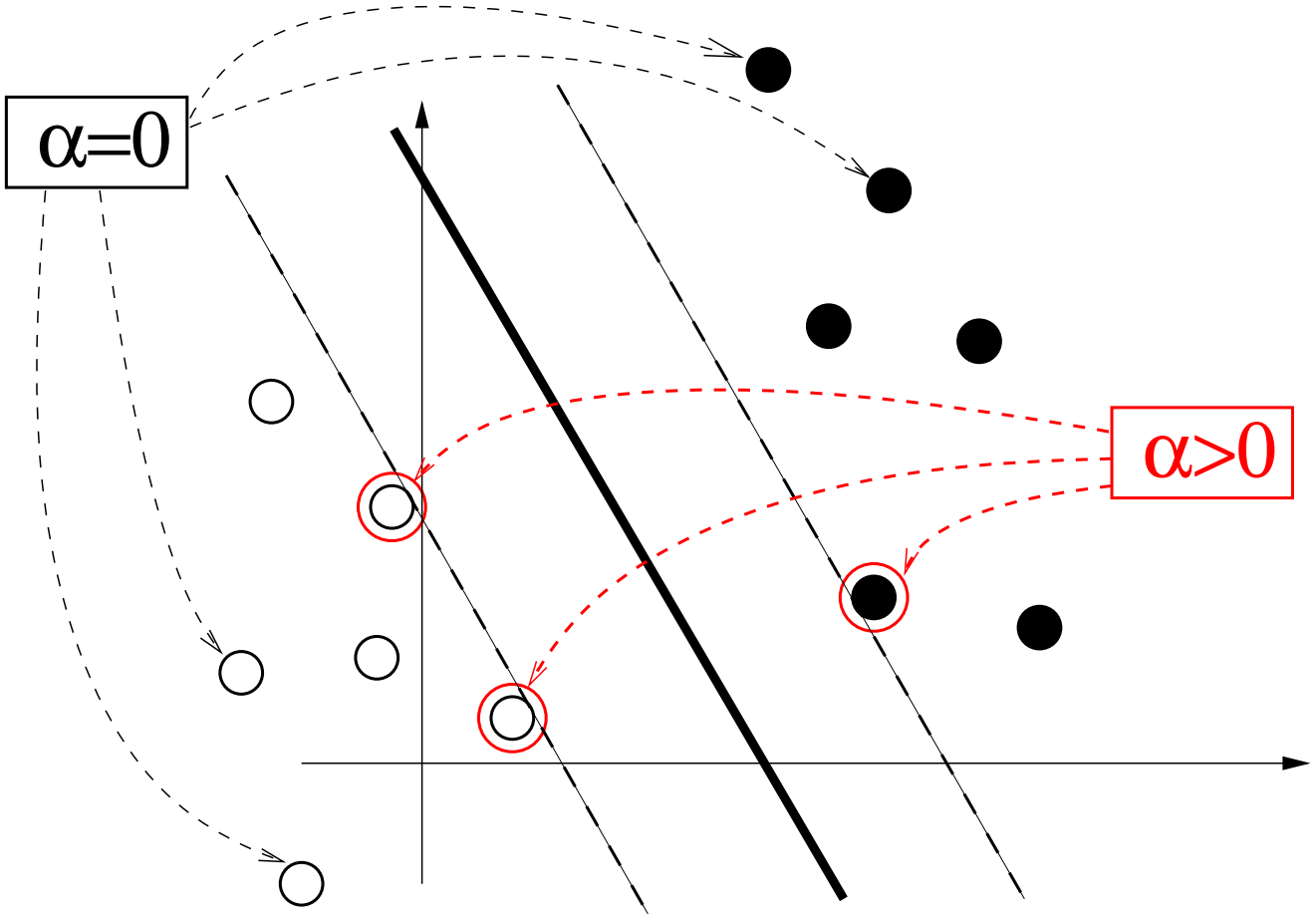
Once $\vec{\alpha}^*$ is found, we recover $(\vec{w}^*, b^*)$ corresponding to the optimal hyperplane. $w^*$ is given by:

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

and the decision function is therefore:

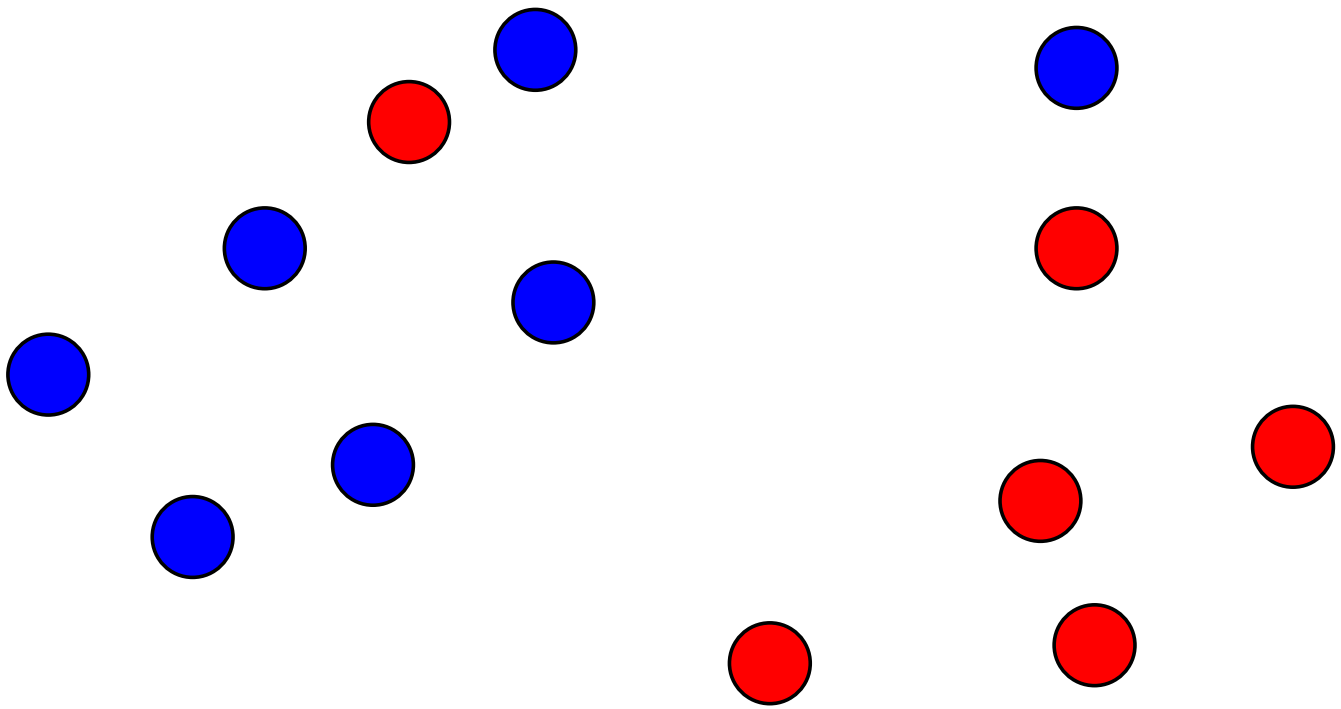
$$\begin{aligned} f^*(\vec{x}) &= \vec{w}^*.\vec{x} + b^* \\ &= \sum_{i=1}^{n} \alpha_i \vec{x}_i.\vec{x} + b^* \,. \end{aligned} \tag{1}$$

# Interpretation: support vectors

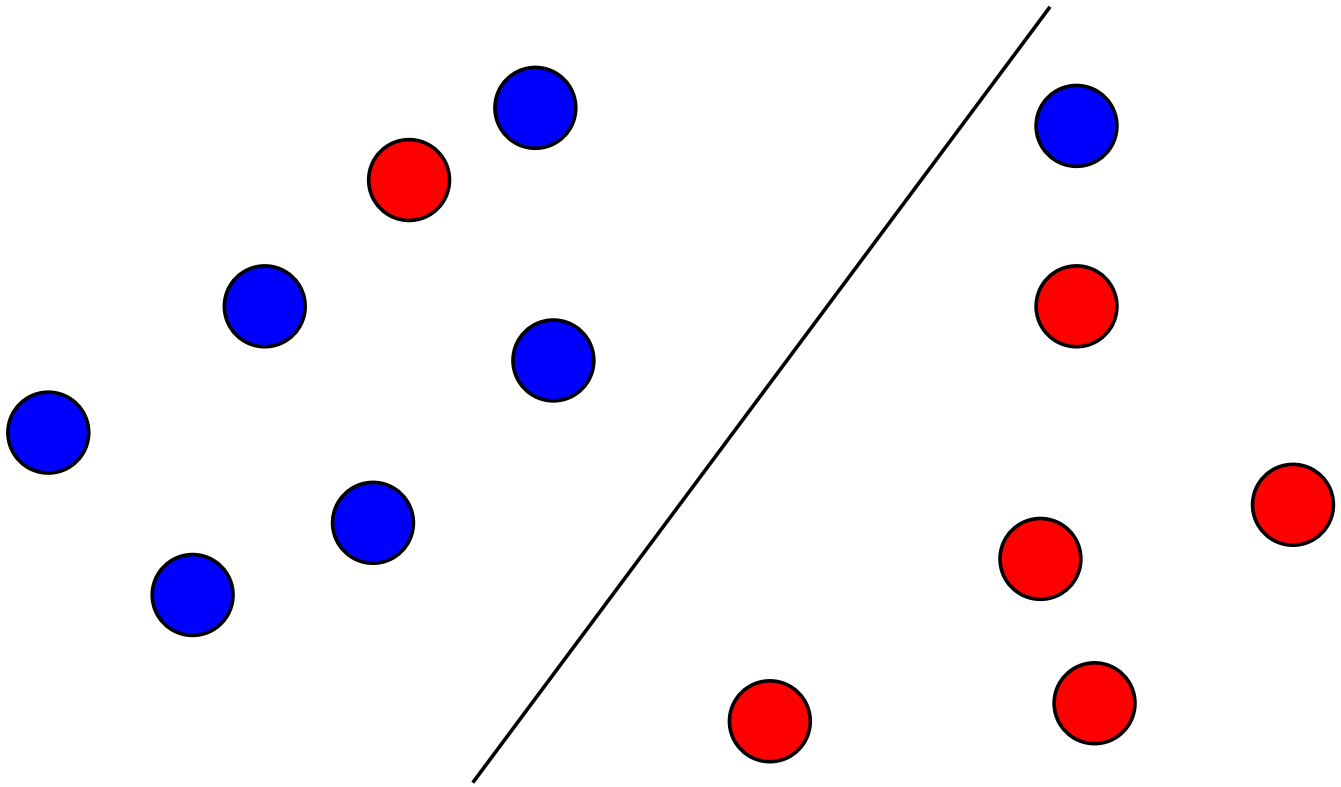


# Soft-Margin Linear Support Vector Machines

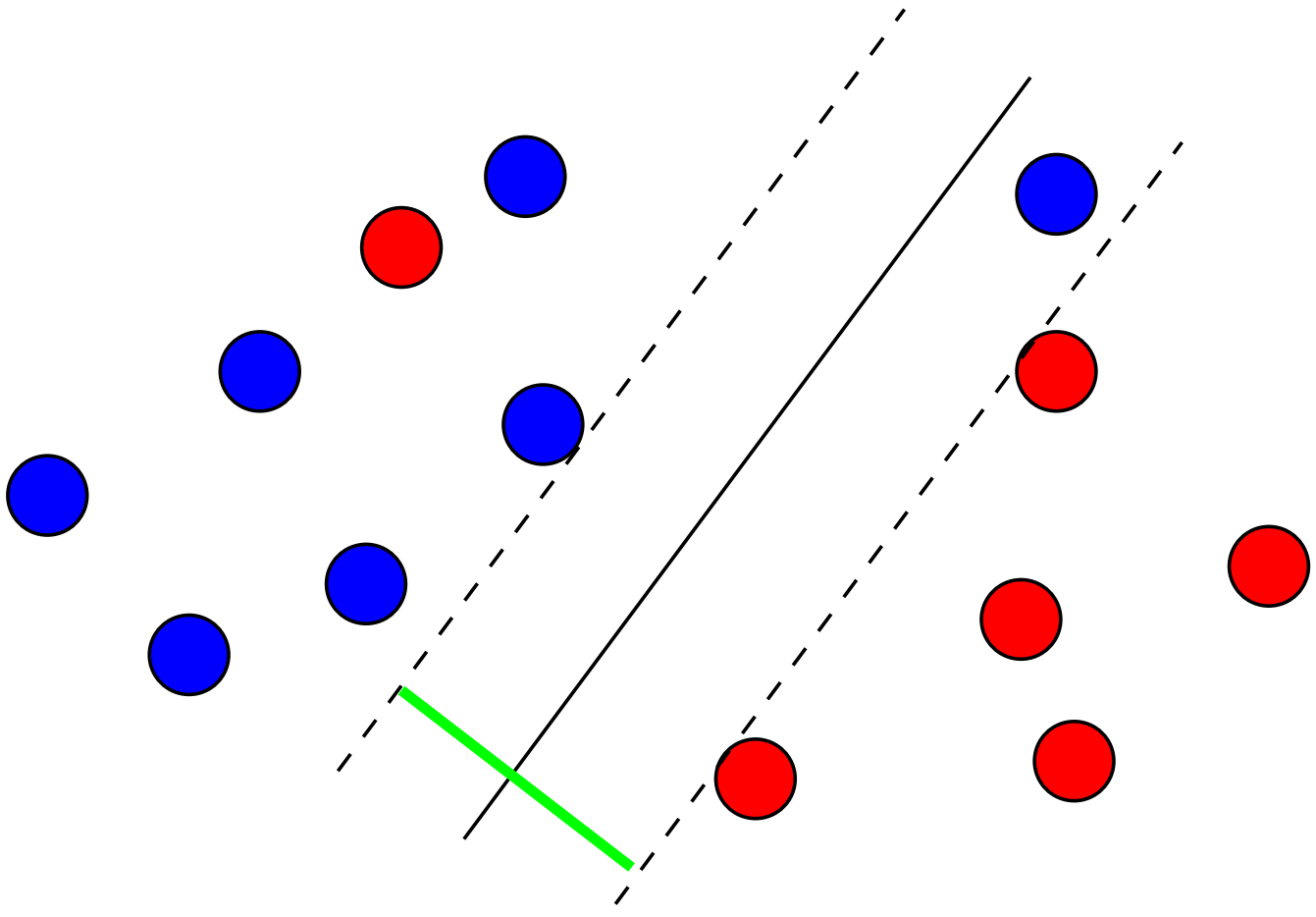
# What if data are not linearly separable?



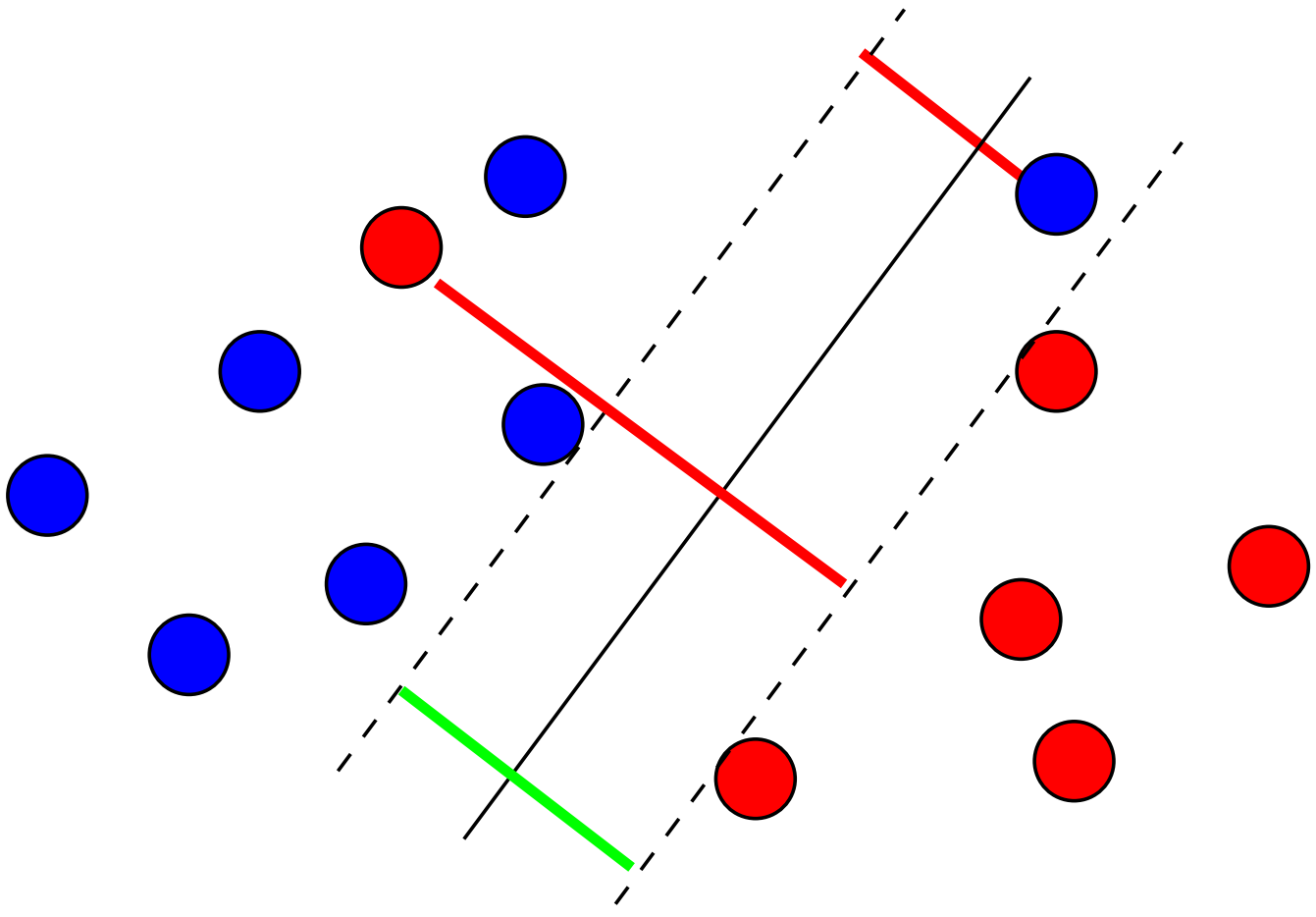
# What if data are not linearly separable?



# What if data are not linearly separable?



# What if data are not linearly separable?

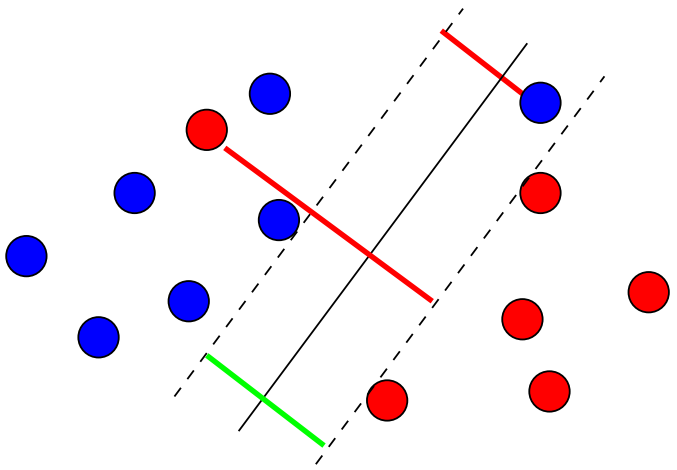


# Soft-margin SVM

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

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

- $C$ is a parameter



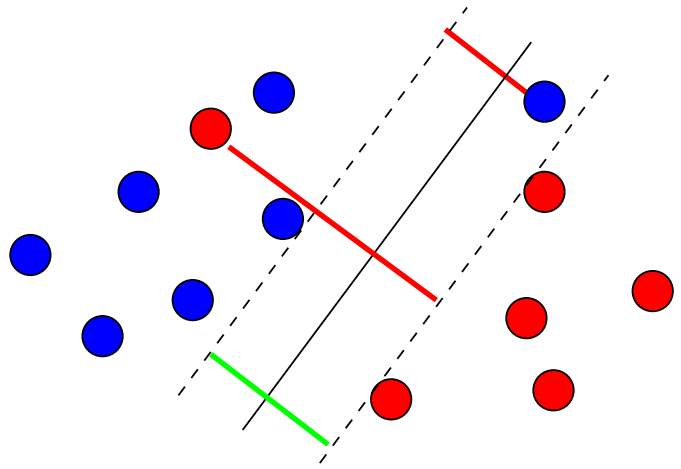
# Soft-margin SVM formulation

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

$$margin(\vec{x}, y) = y\left(\vec{w}.\vec{x} + b\right)$$

- The error is
  - 0 if $margin(\vec{x}, y) > 1$,
  - $1 - margin(\vec{x}, y)$ otherwise.
- The soft margin SVM solves:

$$\min_{\vec{w},b} \left\{ ||\vec{w}||^2 + C \sum_{i=1}^{n} \max\left(0, 1 - y_i\left(\vec{w}.\vec{x}_i + b\right)\right) \right\}$$

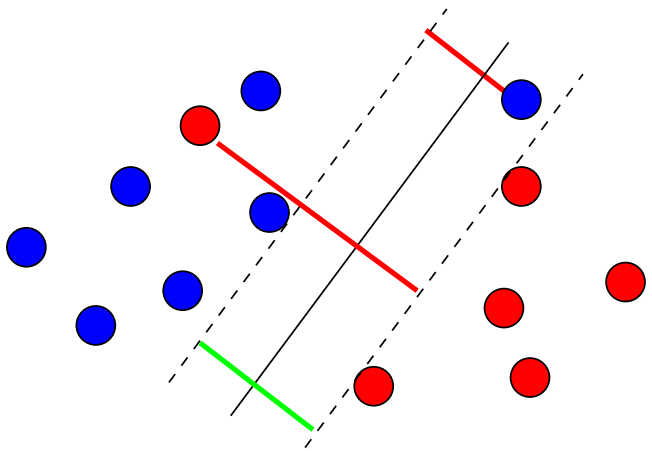


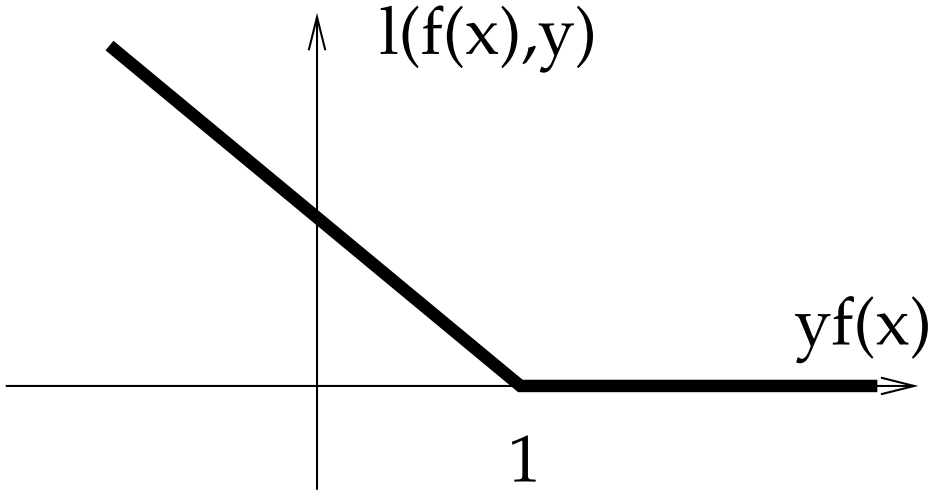
# Soft-margin SVM and hinge loss

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

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

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





# Dual formulation of soft-margin SVM (*exercice*)

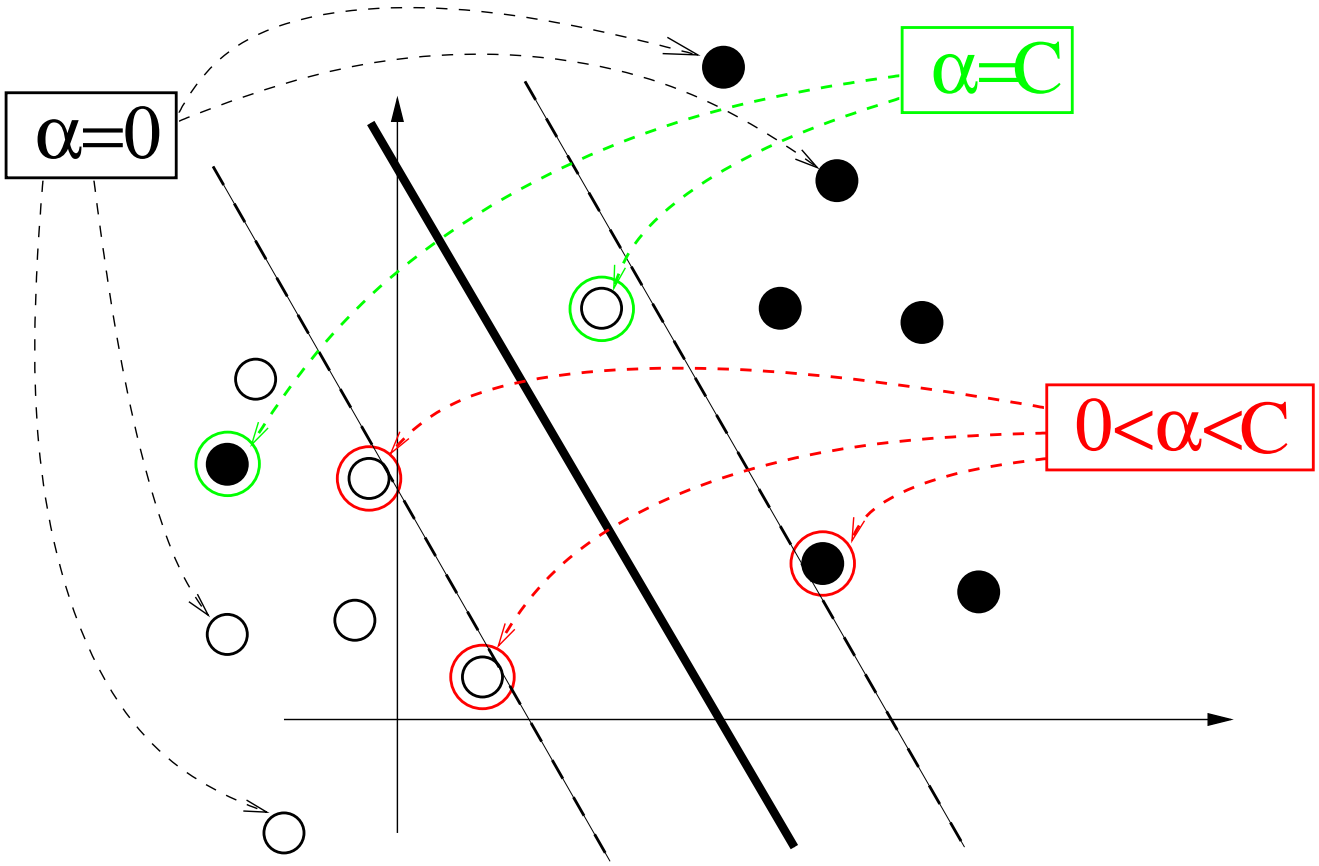
Maximize

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

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



# Primal (for large $n$) vs dual (for large $p$) optimization

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

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

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

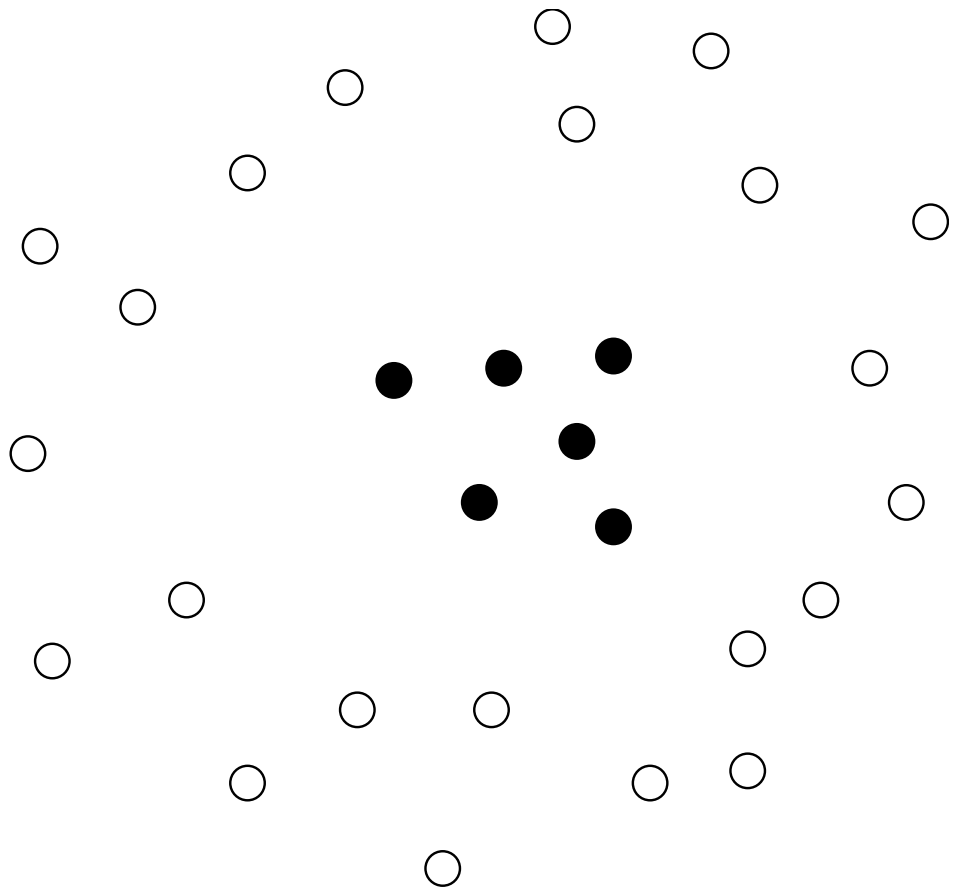
$$L(\vec{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \vec{x}_i . \vec{x}_j \, ,$$

under the constraints:

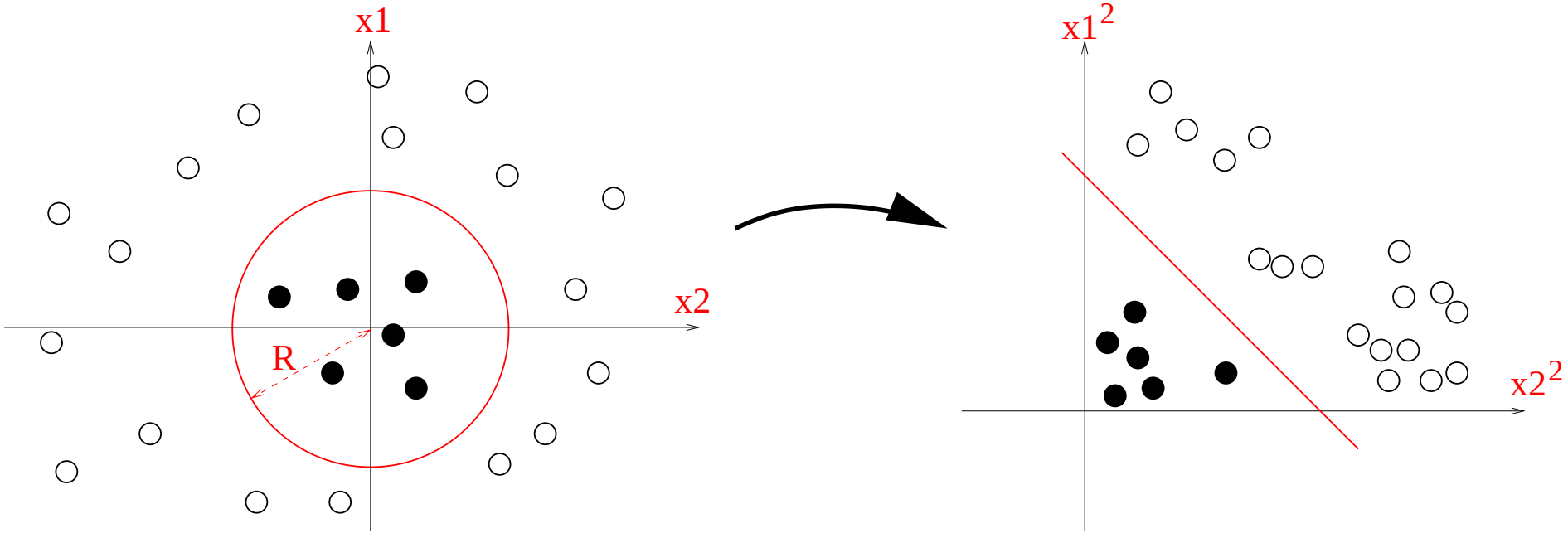
$$\begin{cases} 0 \leq \alpha_i \leq C, \quad \text{for } i = 1, \ldots, n \\ \sum_{i=1}^{n} \alpha_i y_i = 0. \end{cases}$$

# Non-Linear Support Vector Machines and Kernels

# Sometimes linear methods are not interesting



# Solution: nonlinear mapping to a feature space




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

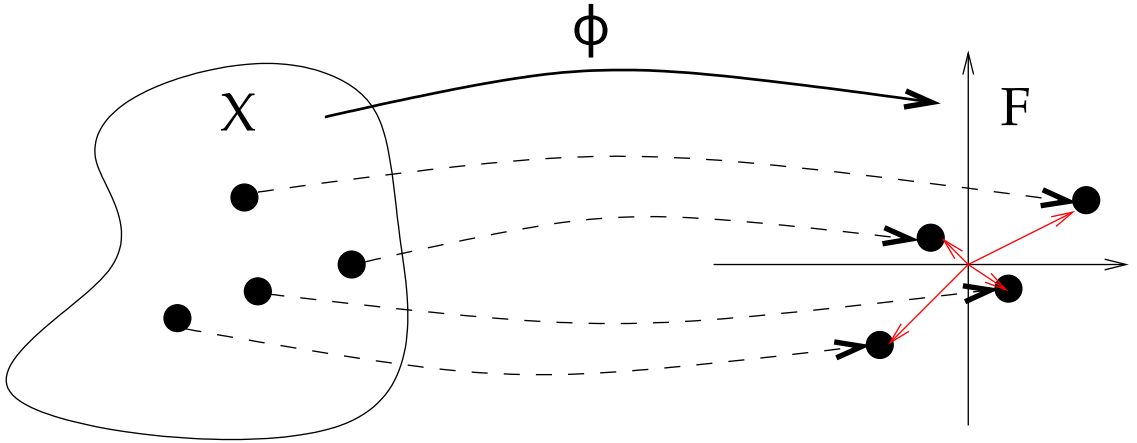
## Definition

For a given mapping

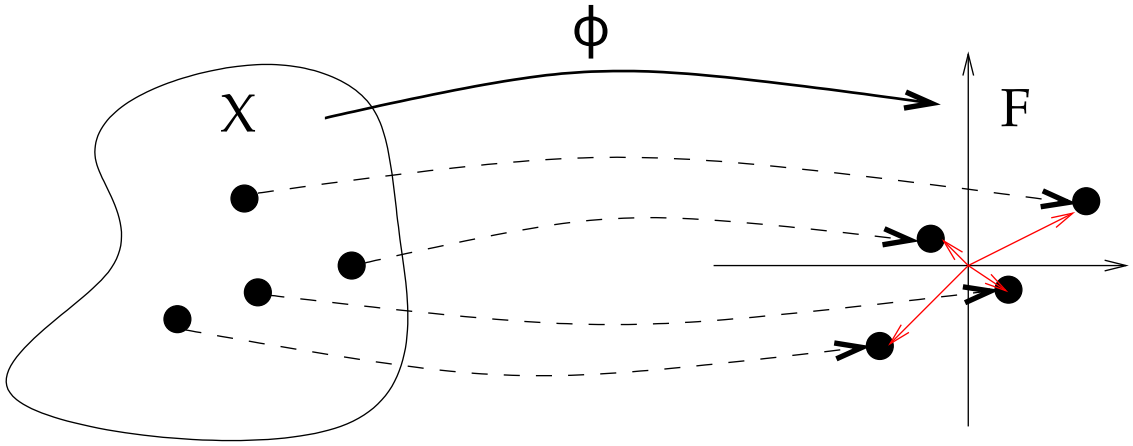
$$\Phi : \mathcal{X} \mapsto \mathcal{H}$$

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

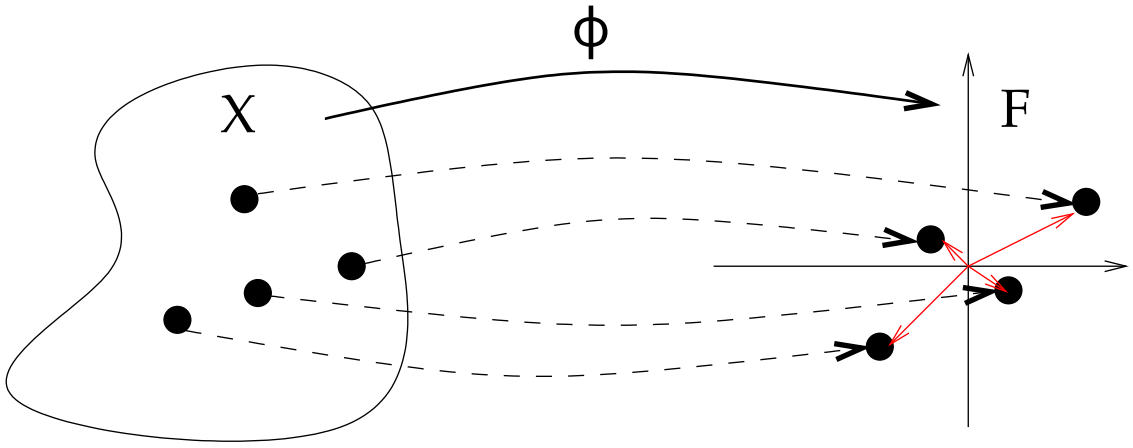



Let $\mathcal{X} = \mathcal{H} = \mathbb{R}^2$ and for $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ let $\Phi(x) = \begin{pmatrix} x_1^2 \\ x_2^2 \end{pmatrix}$

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:

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

- Predict with the decision function

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i x_i^\top x + b^* \,.$$

# 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

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

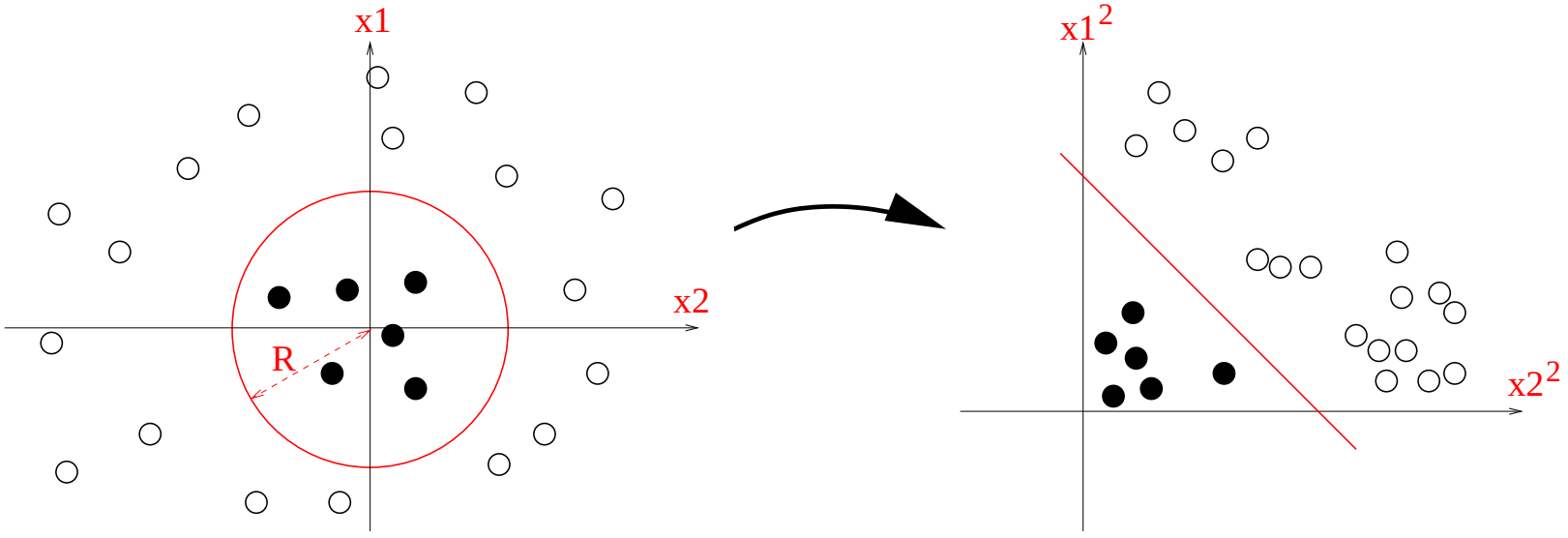
  under the constraints:

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

- Predict with the decision function

$$f(x) = \sum_{i=1}^{n} \alpha_i K\left(x_i, x\right) + b^* \,.$$

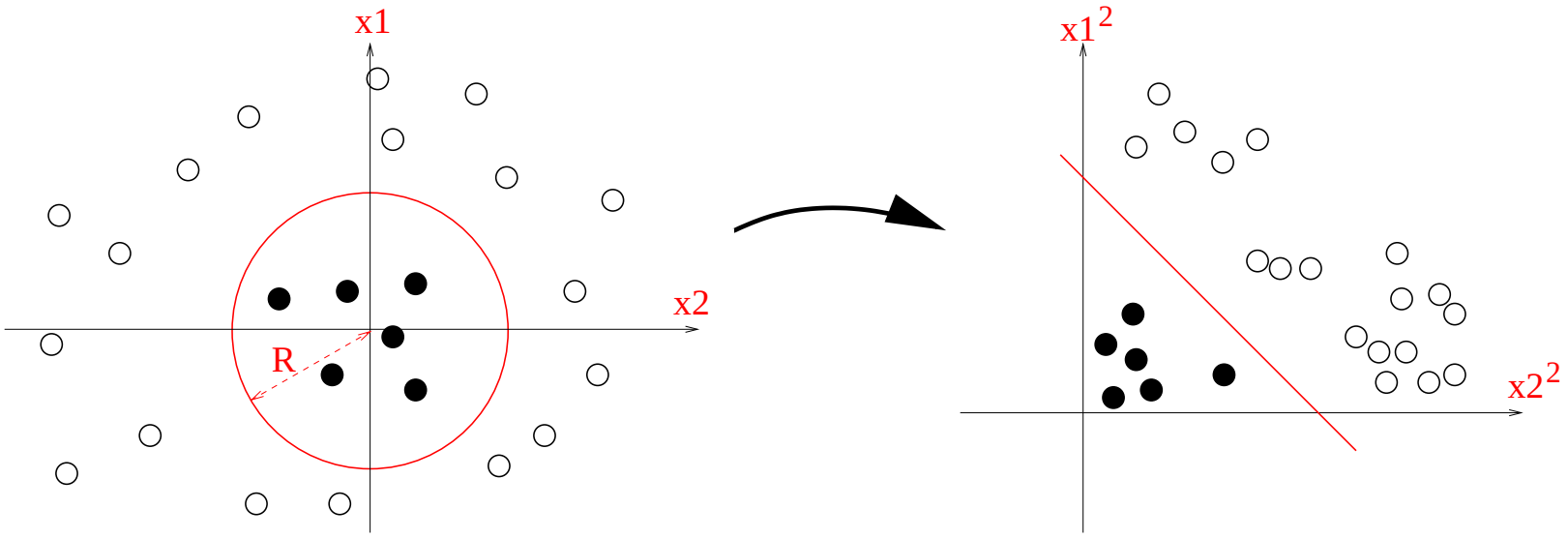
# Trick 2 illustration: polynomial kernel




For $x = (x_1, x_2)^\top \in \mathbb{R}^2$, let $\Phi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \in \mathbb{R}^3$:

$$\begin{aligned} K(x, x') &= x_1^2 x_1'^2 + 2x_1x_2x_1'x_2' + x_2^2 x_2'^2 \\ &= \left(x_1x_1' + x_2x_2'\right)^2 \\ &= \left(x^\top x'\right)^2 . \end{aligned}$$

# Trick 2 illustration: polynomial kernel




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 exercice.)*

# Combining tricks: learn a polynomial discrimination rule with SVM

- Train the SVM by maximizing

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

  under the constraints:

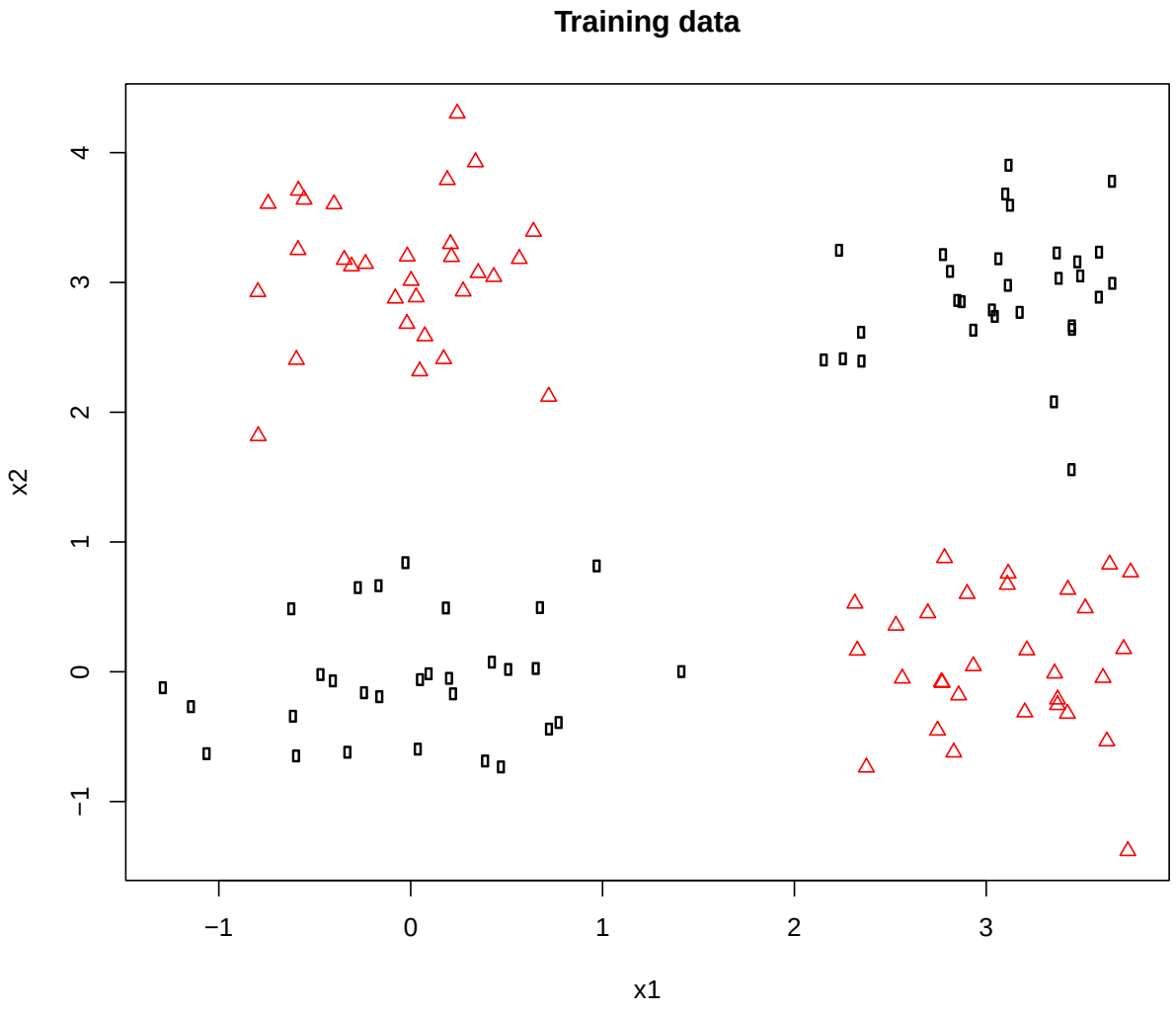
$$\begin{cases} 0 \leq \alpha_i \leq C , & \text{for } i = 1, \ldots, n \\ \sum_{i=1}^{n} \alpha_i y_i = 0 . \end{cases}$$

- Predict with the decision function

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i \left( x_i^\top x + 1 \right)^d + b^* .$$

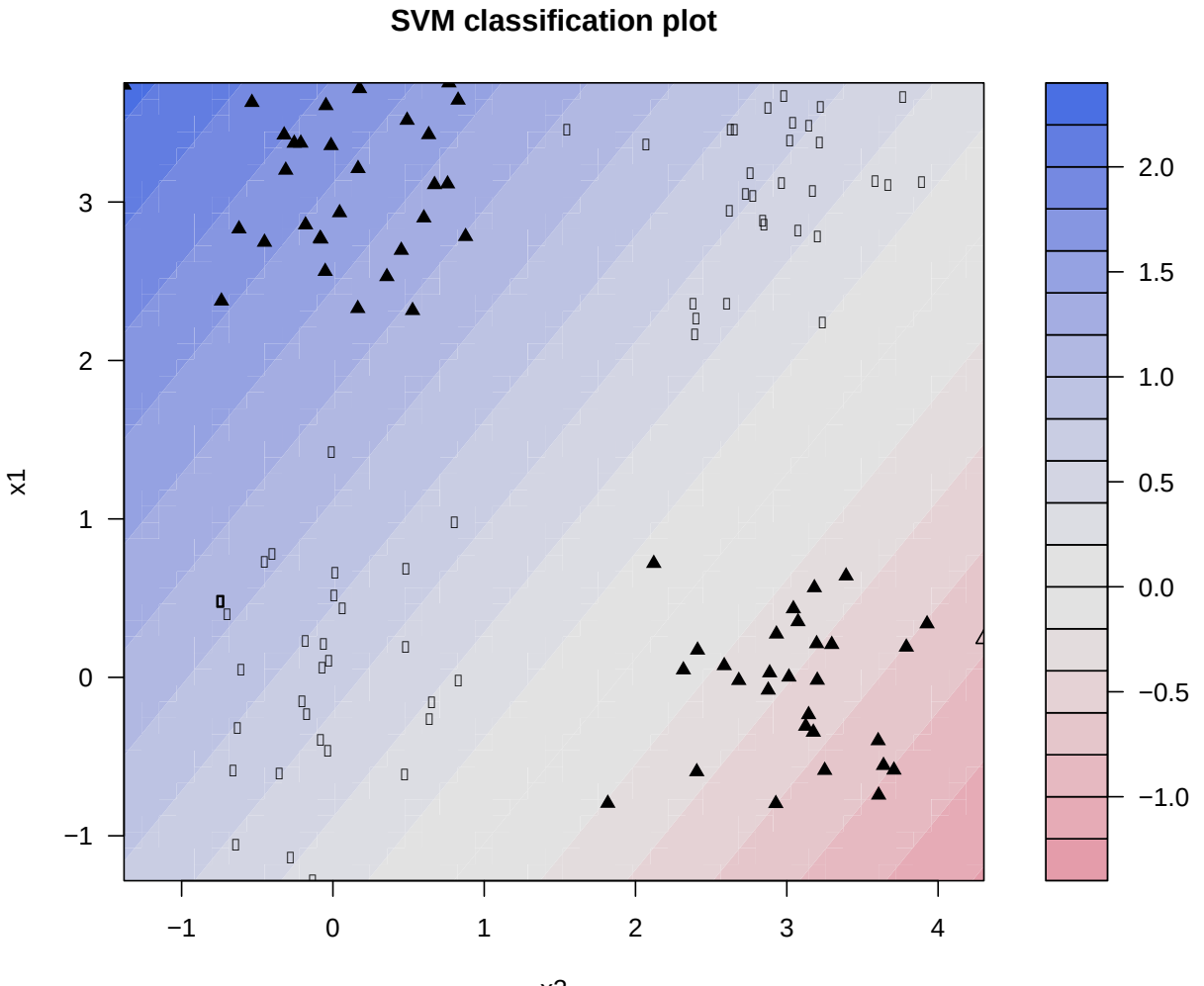
# Illustration: toy nonlinear problem

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



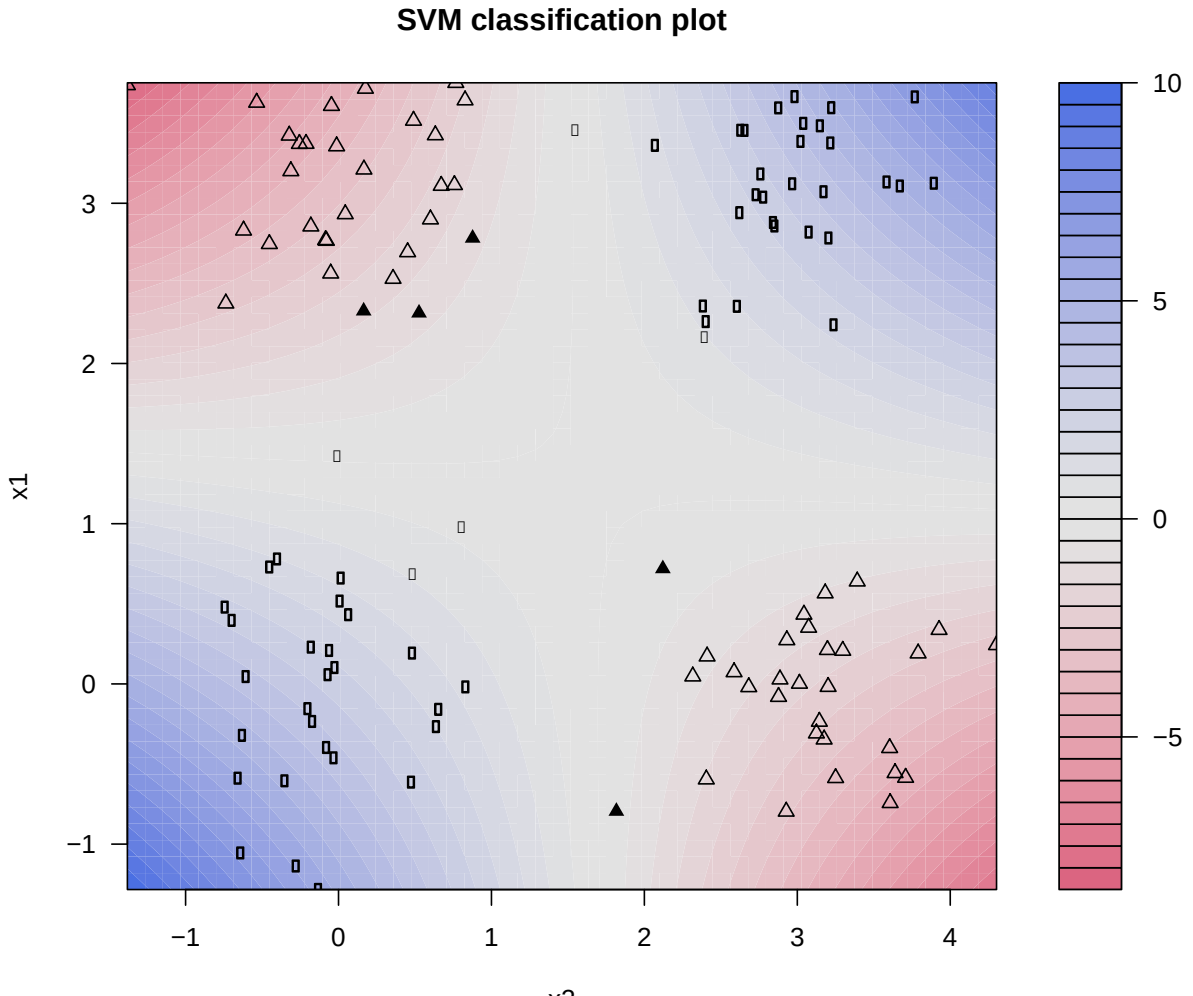
# Illustration: toy nonlinear problem, linear SVM

```
> library(kernlab)
> svp <- ksvm(x,y,type="C-svc",kernel='vanilladot')
> plot(svp,data=x)
```



# Illustration: toy nonlinear problem, polynomial SVM

```
> svp <- ksvm(x,y,type="C-svc", ...
                          kernel=polydot(degree=2))
> plot(svp,data=x)
```



# Which functions $K(x, x')$ are kernels?

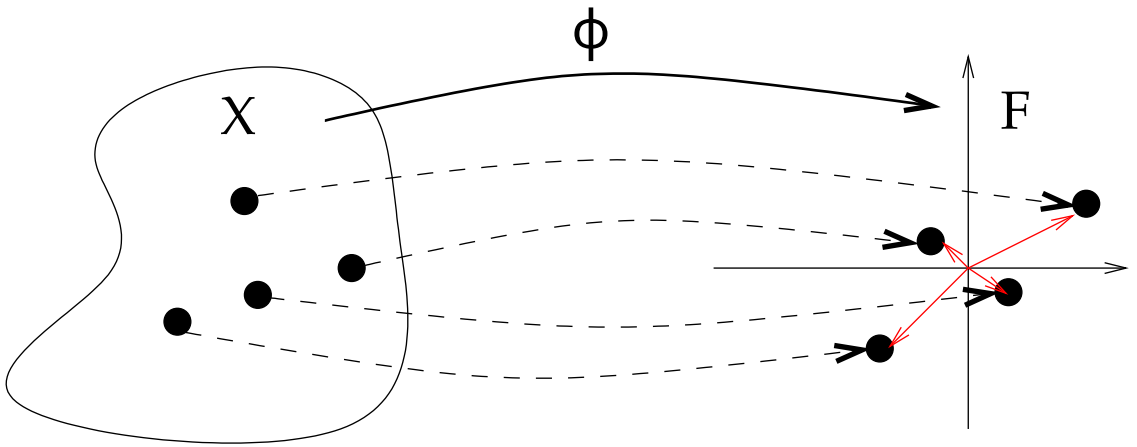
## Definition

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 $\mathcal{X}$:

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



# Positive Definite (p.d.) functions

## 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:

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

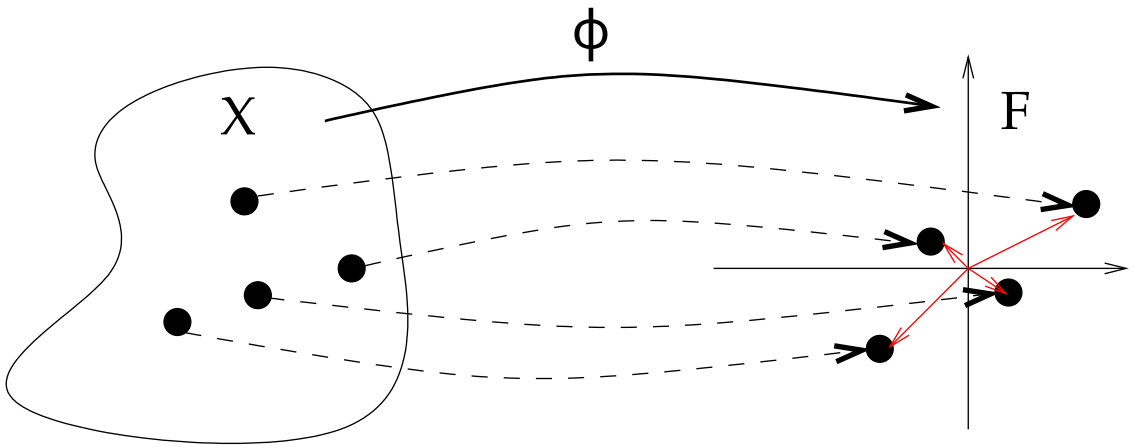
and which satisfies, for all $N \in \mathbb{N}$, $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) \in \mathcal{X}^N$ et $(a_1, a_2, \ldots, a_N) \in \mathbb{R}^N$:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j K\left(\mathbf{x}_i, \mathbf{x}_j\right) \geq 0.$$

# Kernels are p.d. functions

**Theorem (Aronszajn, 1950)**

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



# Proof?

- Kernel $\Longrightarrow$ p.d. function:
  - $\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle_{\mathbb{R}^d} = \langle \Phi(\mathbf{x}'), \Phi(\mathbf{x})_{\mathbb{R}^d} \rangle$ ,
  - $\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle_{\mathbb{R}^d} = \| \sum_{i=1}^{N} a_i \Phi(\mathbf{x}_i) \|^2_{\mathbb{R}^d} \geq 0$ .
- P.d. function $\Longrightarrow$ kernel: more difficult...

# Kernel examples

- **Polynomial** $\mathcal{X} = \mathbb{R}^m$

$$K(u, v) = (u.v + c)^d$$

- **Gaussian Radial Basis Function (RBF)** $\mathcal{X} = \mathbb{R}^m$

$$K(u, v) = \exp\left(-\frac{|u - v|^2}{2\sigma^2}\right)$$

- **Min** $\mathcal{X} = \mathbb{R}_+$

$$K(u, v) = \min(u, v)$$

# Example: SVM with a Gaussian kernel

- Training:

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

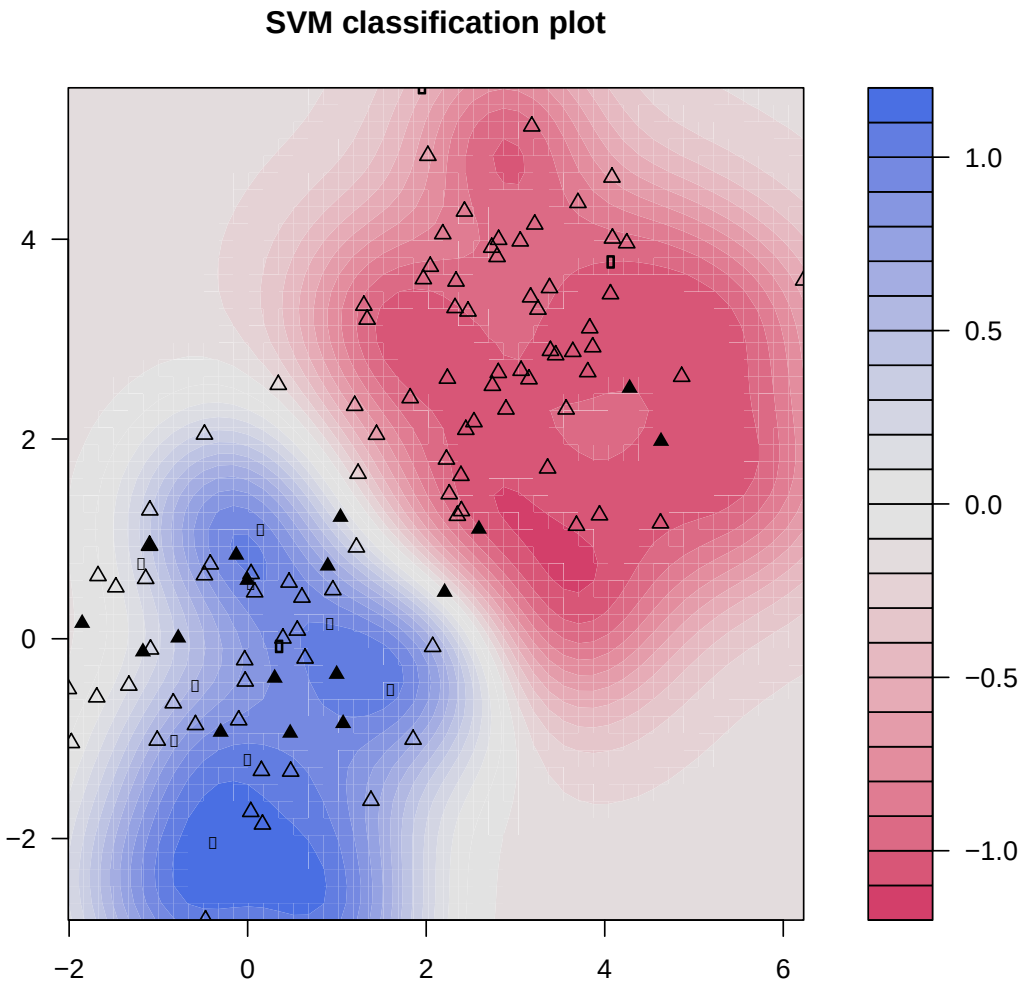
$$\text{s.t. } 0 \leq \alpha_i \leq C, \quad \text{and } \sum_{i=1}^{n} \alpha_i y_i = 0.$$

- Prediction

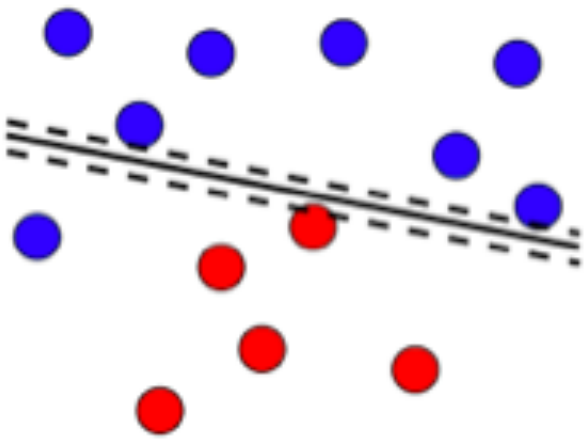
$$f(\vec{x}) = \sum_{i=1}^{n} \alpha_i \exp\left( -\frac{||\vec{x} - \vec{x}_i||^2}{2\sigma^2} \right)$$

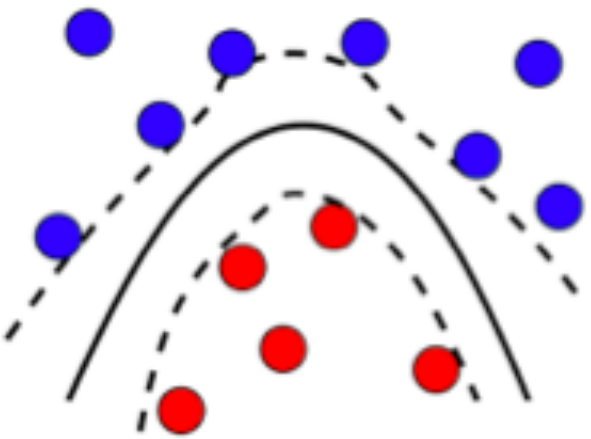
# Example: SVM with a Gaussian kernel

$$f(\vec{x}) = \sum_{i=1}^{n} \alpha_i \exp\left(-\frac{||\vec{x} - \vec{x}_i||^2}{2\sigma^2}\right)$$

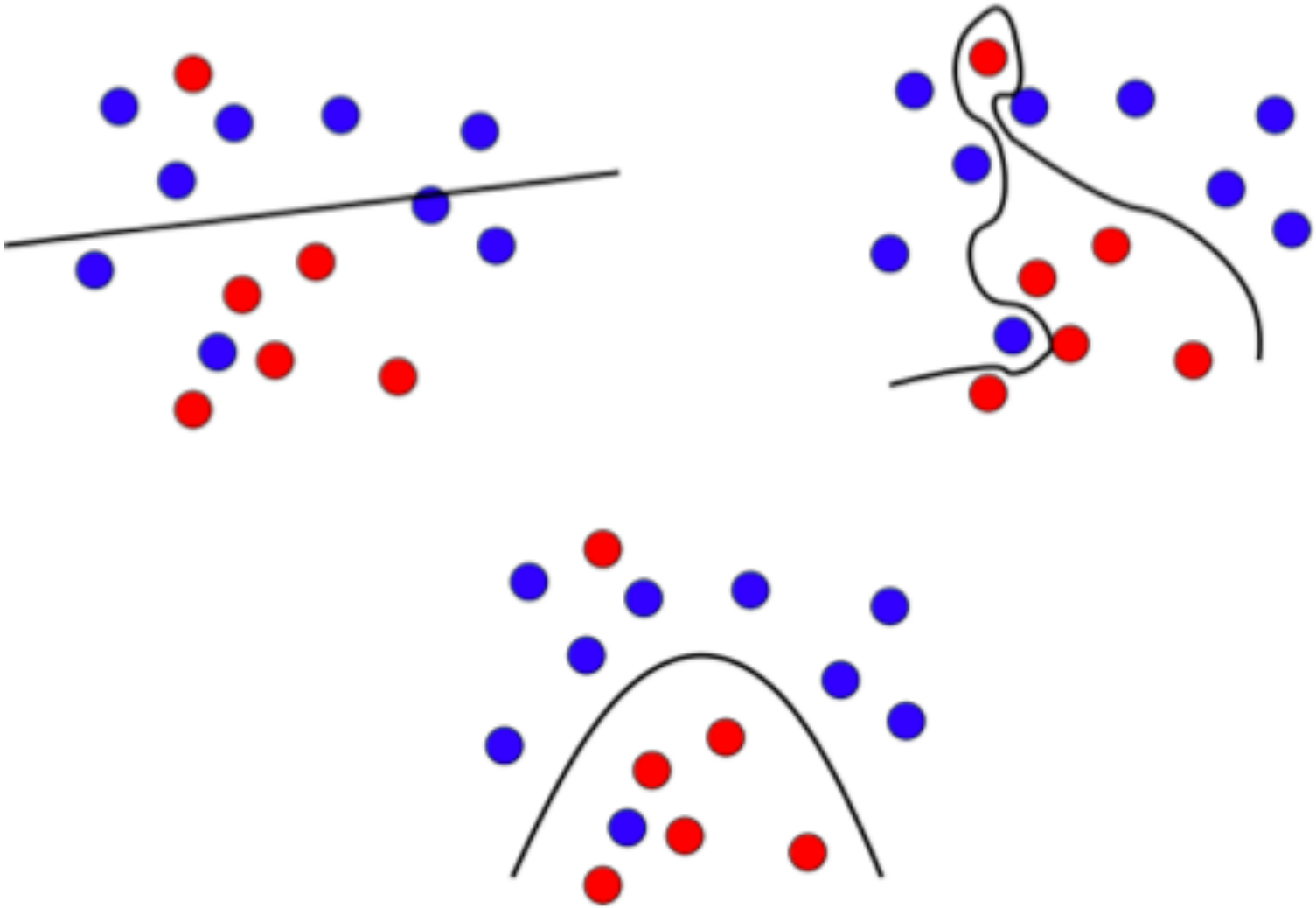


# Linear vs nonlinear SVM





# Regularity vs data fitting trade-off

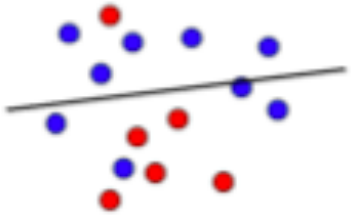


# $C$ controls the trade-off

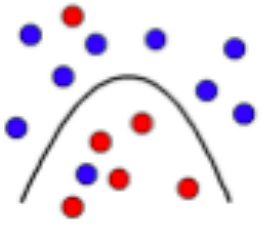
$$\min_f \left\{ \frac{1}{margin(f)} + C \times errors(f) \right\}$$

- Large C :
  - makes few errors

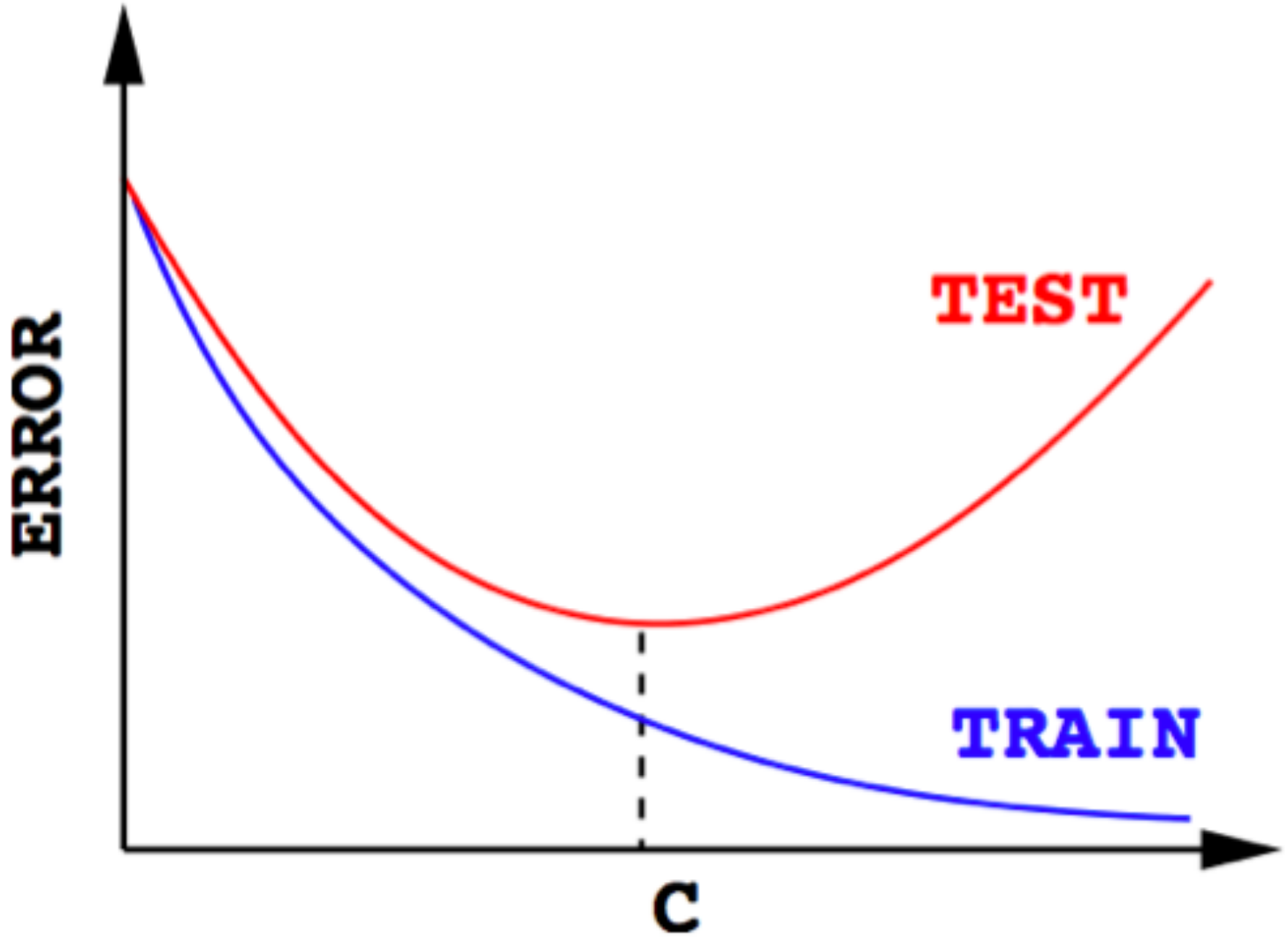
- Small C :
  - ensure a large margin



- 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)

# Kernels for Strings

# Supervised sequence classification

## Data (training)

- Secreted proteins:

```
MASKATLLLAFTLLFATCIARHQQRQQQQNQCQLQNIEA...
MARSSLFTFLCLAVFINGCLSQIEQQSPWEFQGSEVW...
MALHTVLIMLSLLPMLEAQNPEHANITIGEPITNETLGWL...
...
```

- Non-secreted proteins:

```
MAPPSVFAEVPQAQPVLVFKLIADFREDPDPRKVNLGVG...
MAHTLGLTQPNSTEPHKISFTAKEIDVIEWKGDILVVG...
MSISESYAKEIKTAFRQFTDFPIEGEQFEDFLPIIGNP..
...
```

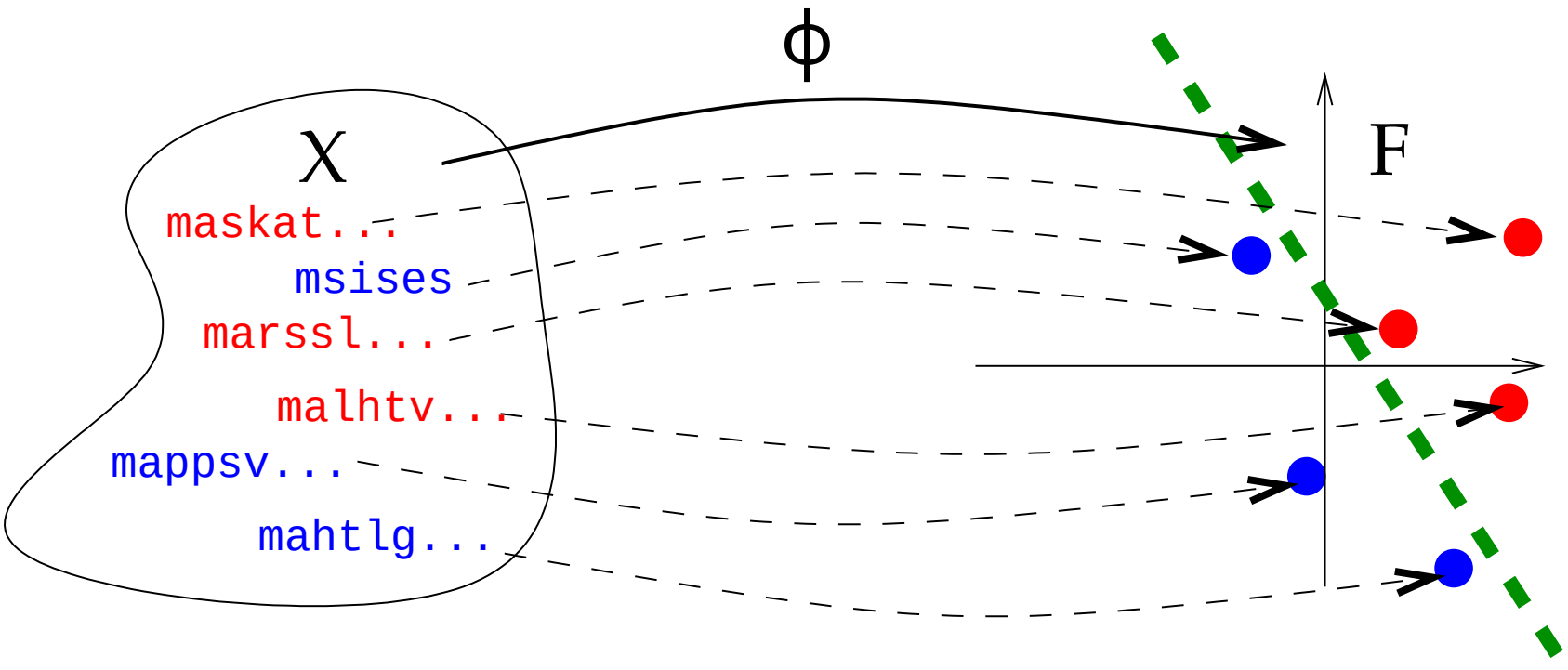
## Goal

- 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...)



# Example: substring indexation

## The approach

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

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

where $\Phi_u(\mathbf{x})$ can be:

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

# Spectrum kernel (1/2)

## Kernel definition

- The 3-spectrum of

$$\mathbf{x} = \texttt{CGGSLIAMMWFGV}$$

is:

$$(\texttt{CGG}, \texttt{GGS}, \texttt{GSL}, \texttt{SLI}, \texttt{LIA}, \texttt{IAM}, \texttt{AMM}, \texttt{MMW}, \texttt{MWF}, \texttt{WFG}, \texttt{FGV}) \ .$$

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

$$K(\mathbf{x}, \mathbf{x}') := \sum_{u \in \mathcal{A}^k} \Phi_u(\mathbf{x}) \Phi_u(\mathbf{x}') \ .$$

# Spectrum kernel (2/2)

## Implementation

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

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

## 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 alignmnent kernel (Saigo et al., 2004)

```
CGGSLIAMM----WFGV
|...|||||....||||
C---LIVMMNRLMWFGV
```

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

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

$$K_{LA}^{(\beta)}(x,y) = \sum_{\pi \in \Pi(x,y)} \exp\left(\beta s_{S,g}(x,y,\pi)\right) \quad \text{is a kernel}$$

# LA kernel is p.d.: proof (1/2)

## 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 $K_1 \star K_2$, is defined for any $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ by:

$$K_1 \star K_2(\mathbf{x}, \mathbf{y}) := \sum_{\mathbf{x}_1\mathbf{x}_2=\mathbf{x}, \mathbf{y}_1\mathbf{y}_2=\mathbf{y}} K_1(\mathbf{x}_1, \mathbf{y}_1) K_2(\mathbf{x}_2, \mathbf{y}_2).$$

## Lemma

*If $K_1$ and $K_2$ are p.d. then $K_1 \star K_2$ is p.d..*

# LA kernel is p.d.: proof (2/2)

$$K_{LA}^{(\beta)} = \sum_{n=0}^{\infty} K_0 \star \left( K_a^{(\beta)} \star K_g^{(\beta)} \right)^{(n-1)} \star K_a^{(\beta)} \star K_0 \,,$$

with

- The constant kernel:

$$K_0 (\mathbf{x}, \mathbf{y}) := 1 \,.$$

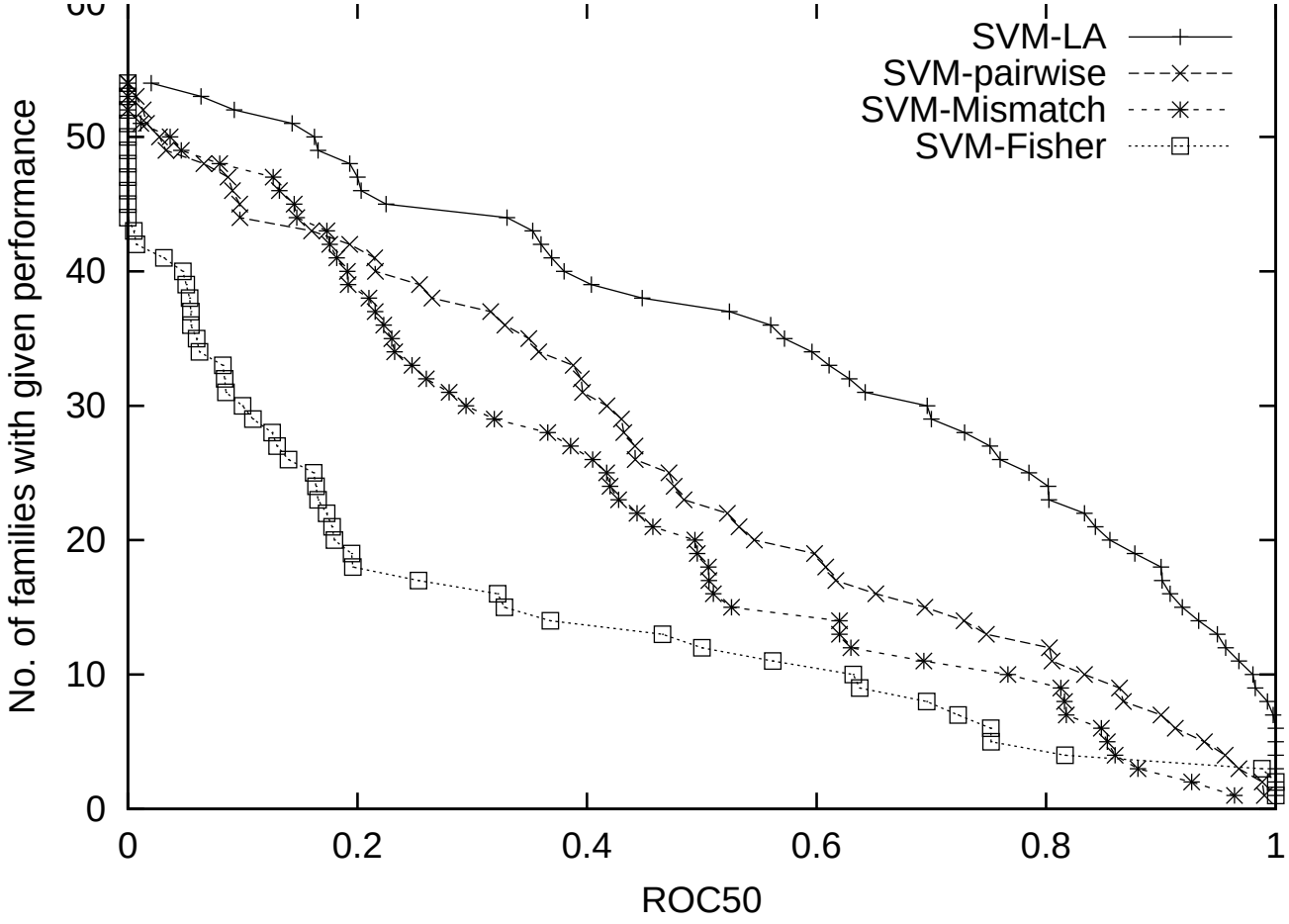
- A kernel for letters:

$$K_a^{(\beta)} (\mathbf{x}, \mathbf{y}) := \begin{cases} 0 & \text{if } |\mathbf{x}| \neq 1 \text{ where } |\mathbf{y}| \neq 1 \,, \\ \exp(\beta S(\mathbf{x}, \mathbf{y})) & \text{otherwise} \,. \end{cases}$$

- A kernel for gaps:

$$K_g^{(\beta)} (\mathbf{x}, \mathbf{y}) = \exp\left[\beta \left( g(|\mathbf{x}|) + g(|\mathbf{x}|) \right)\right] \,.$$

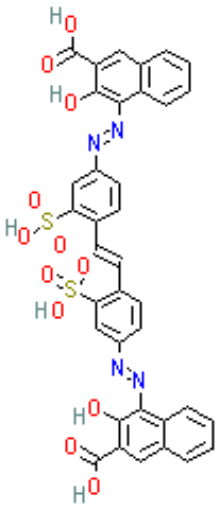
# The choice of kernel matters



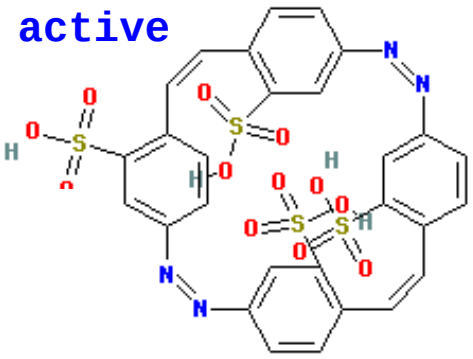

Performance on the SCOP superfamily recognition benchmark (from Saigo et al., 2004).

# Kernels for Graphs

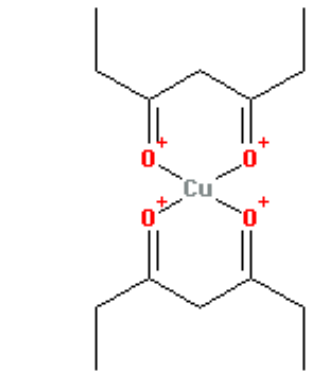
# Virtual screening for drug discovery

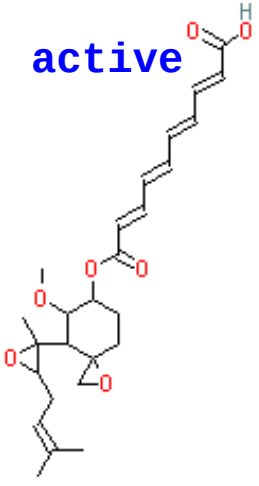


active

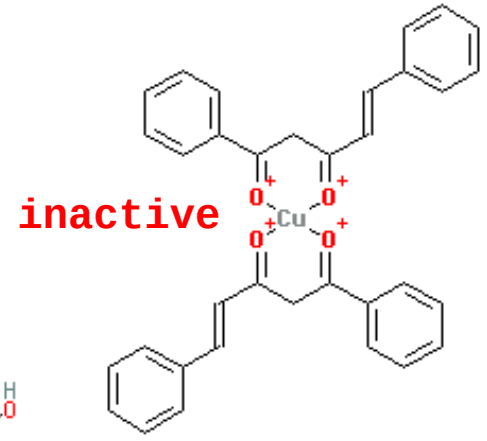


inactive

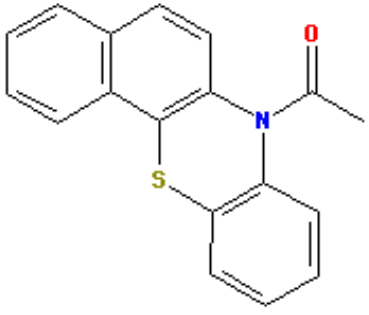




active



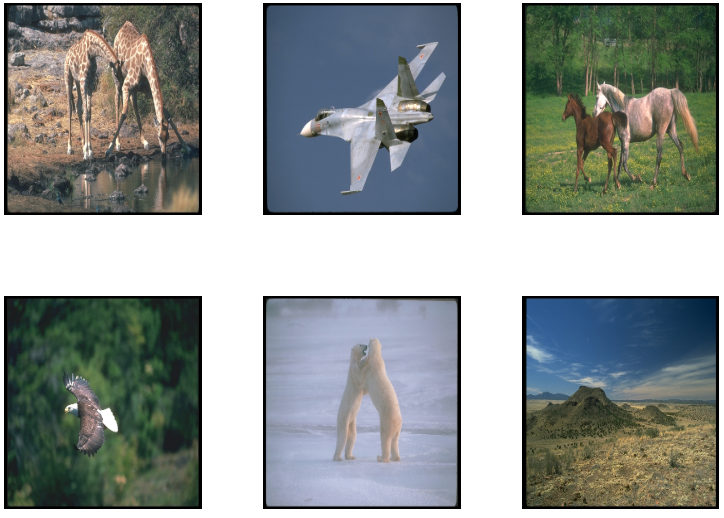
inactive

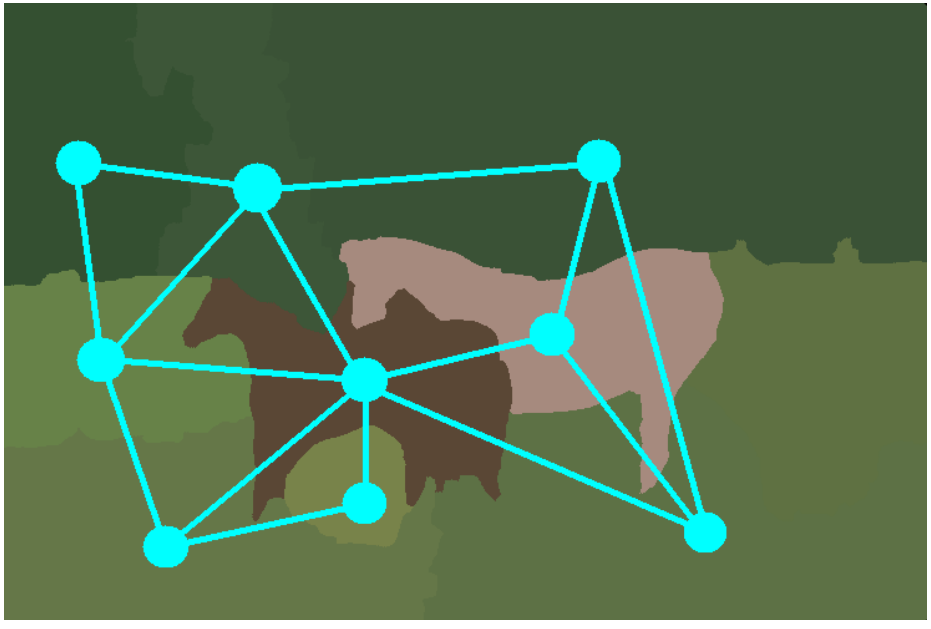


inactive

*NCI AIDS screen results (from http://cactus.nci.nih.gov).*

# Image retrieval and classification





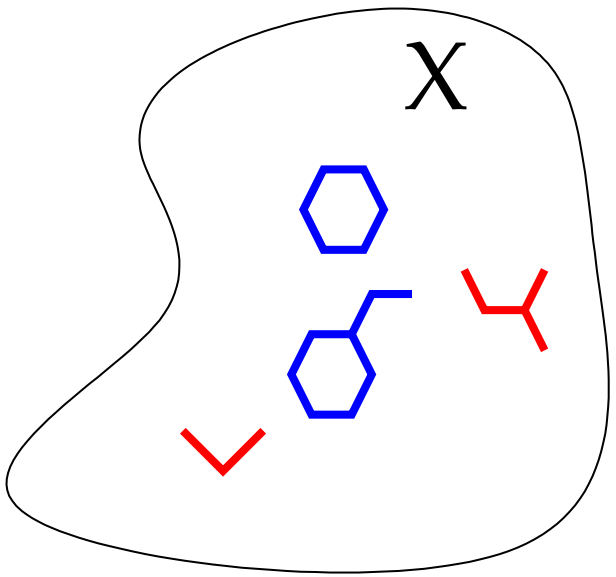
*From Harchaoui and Bach (2007).*

# 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 $\mathcal{H}$.

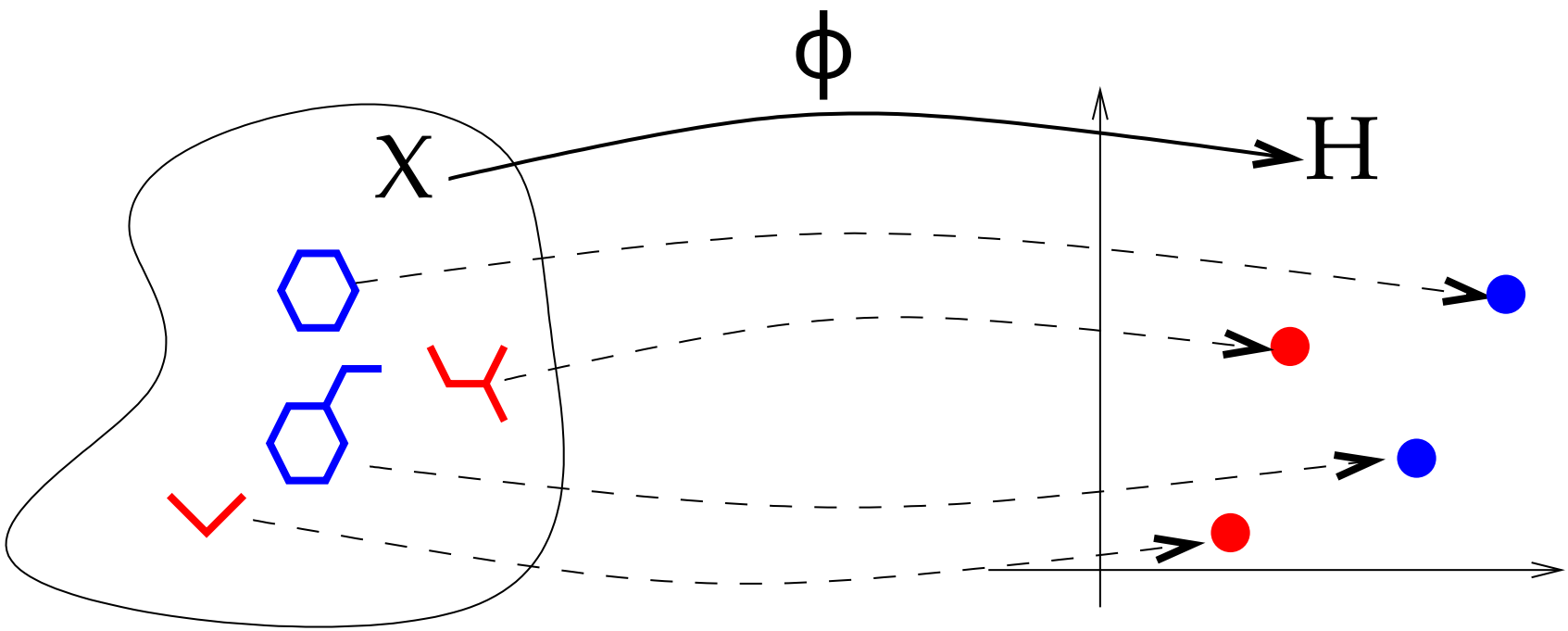


# Graph kernels

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 $\mathcal{H}$.

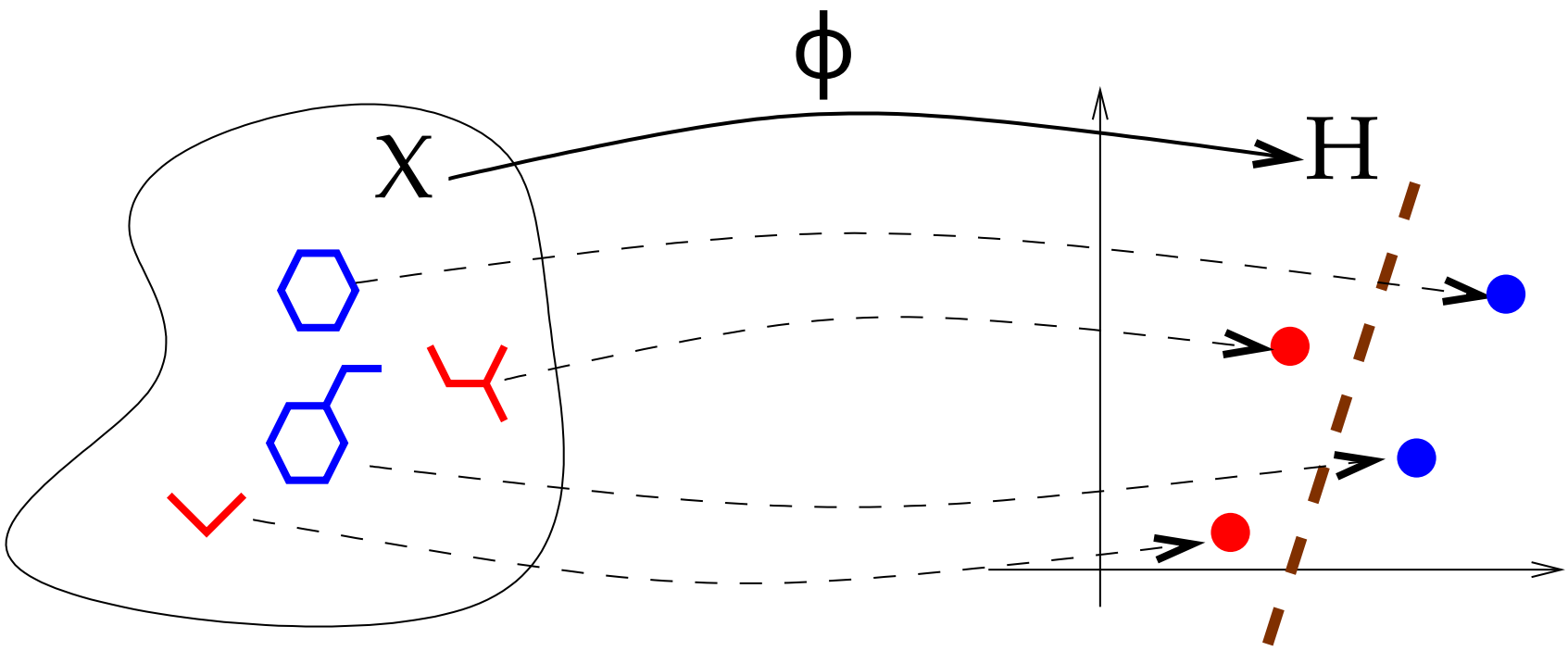


# Graph kernels

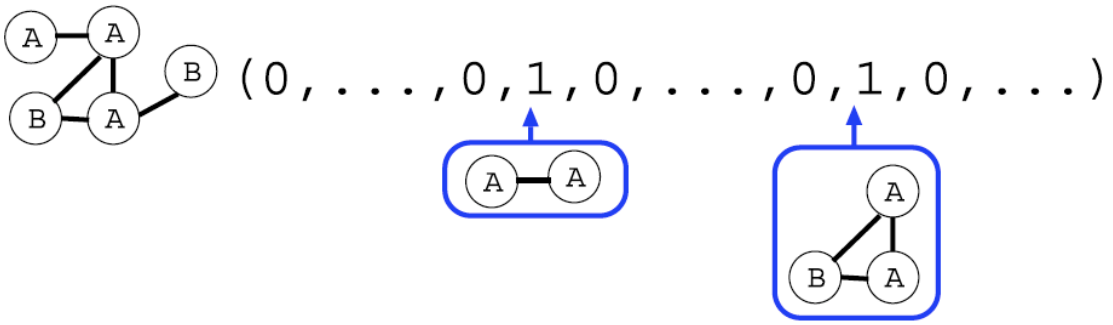
1. Represent each graph $x$ by a vector $\Phi(x) \in \mathcal{H}$, either explicitly or implicitly through the kernel

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

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



# Indexing by all subgraphs?




**Theorem**

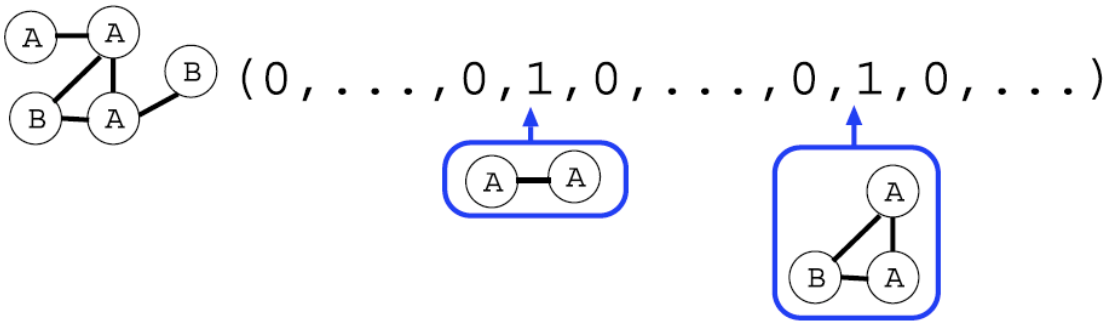
*Computing all subgraph occurrences is NP-hard.*

**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**

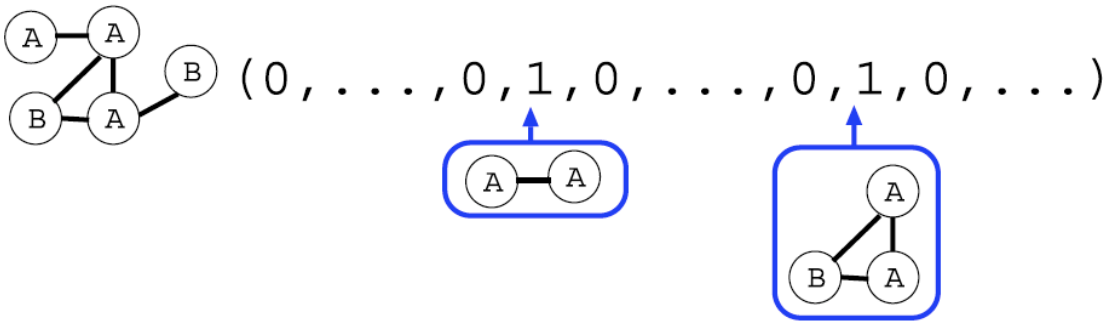
*Computing all subgraph occurrences is NP-hard.*

**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 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.

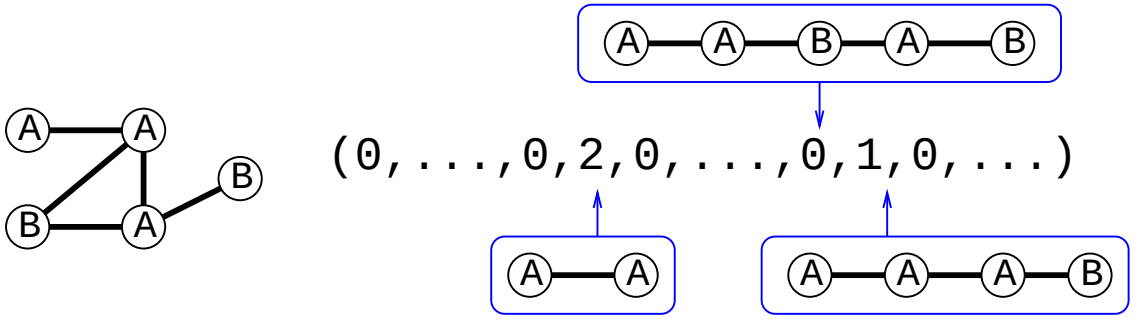
# Indexing by specific subgraphs

## Substructure selection

We can imagine more limited sets of substuctures that lead to more computationnally 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)

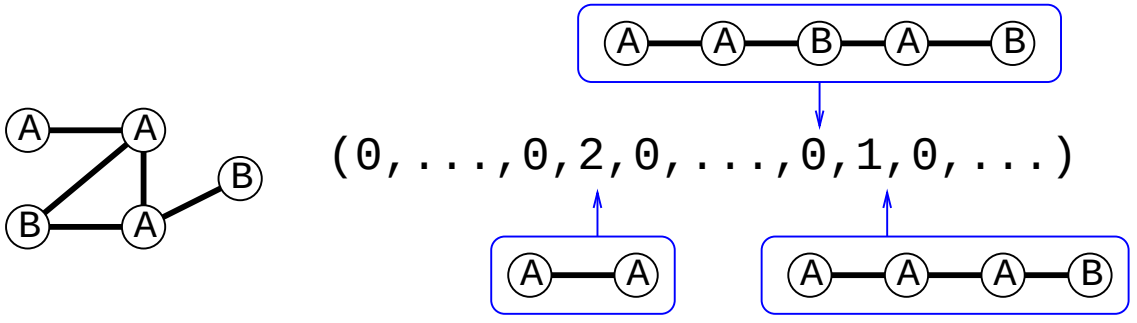
# Example : Indexing by all shortest paths




## 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.

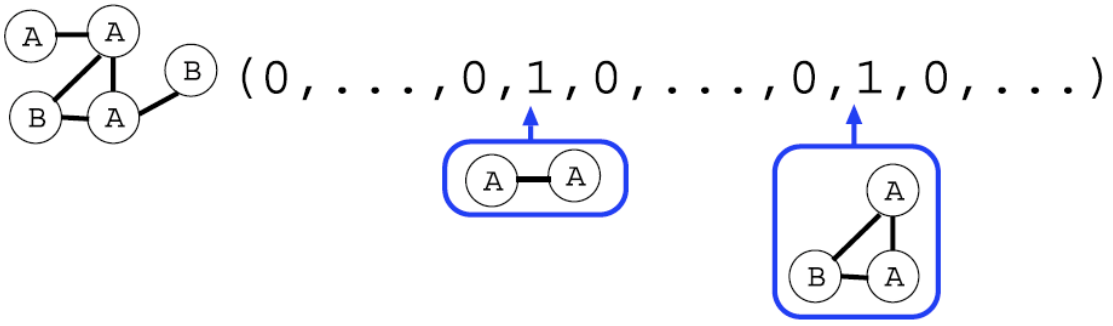
# Example : Indexing by all shortest paths




## 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.

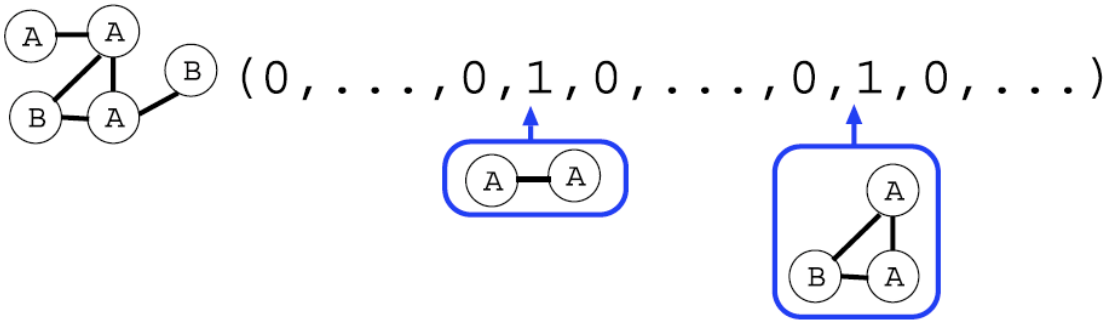
# Example : Indexing by all subgraphs up to $k$ vertices




## 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



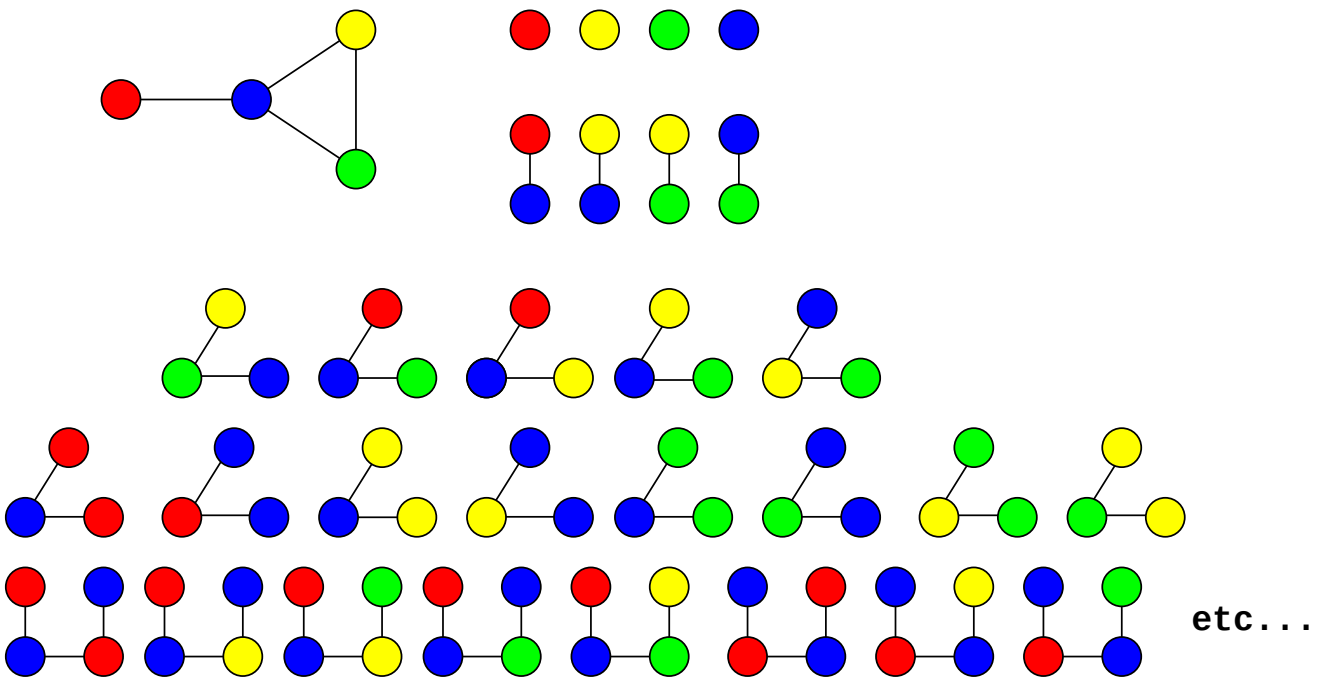

## Properties (Shervashidze et al., 2009)

- 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.

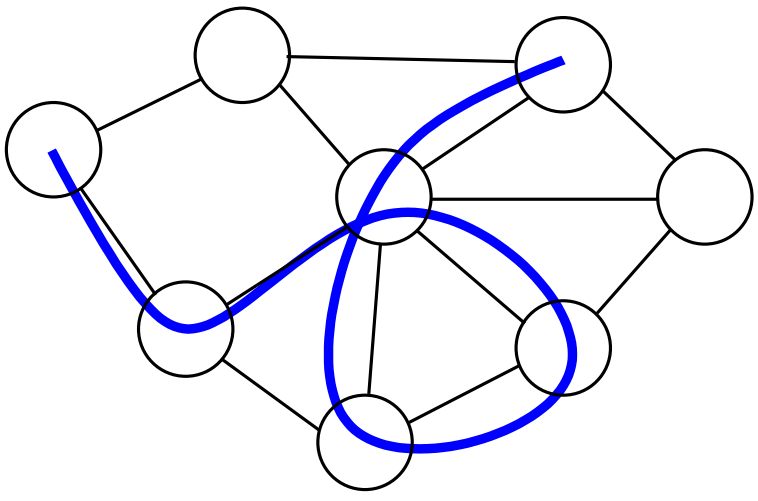
# Walks

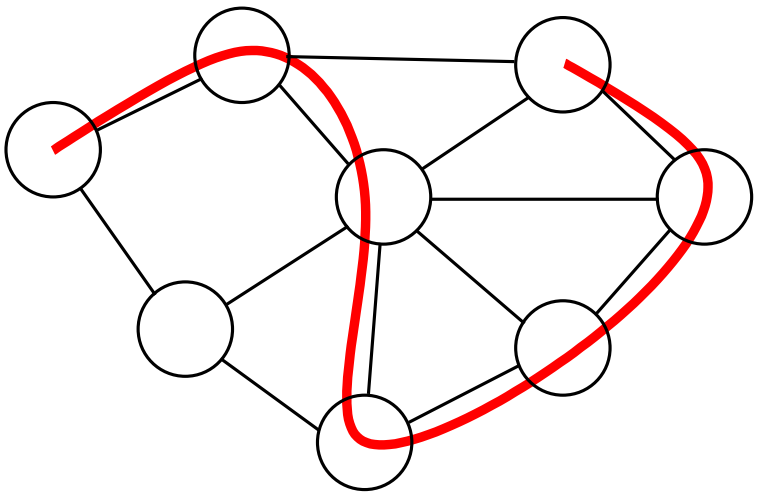
## Definition

- 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 $\mathcal{W}_n(G)$ the set of walks with $n$ vertices of the graph $G$, and $\mathcal{W}(G)$ the set of all walks.



# Walks $\neq$ paths





# Walk kernel

## Definition

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

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

- A walk kernel is a graph kernel defined by:

$$K_{walk}(G_1, G_2) = \sum_{s \in \mathcal{S}} \Phi_s(G_1) \Phi_s(G_2) .$$

# Walk kernel

## Definition

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

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

- A walk kernel is a graph kernel defined by:

$$K_{walk}(G_1, G_2) = \sum_{s \in \mathcal{S}} \Phi_s(G_1) \Phi_s(G_2) .$$

# 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 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 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 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).

# Computation of walk kernels

**Proposition**

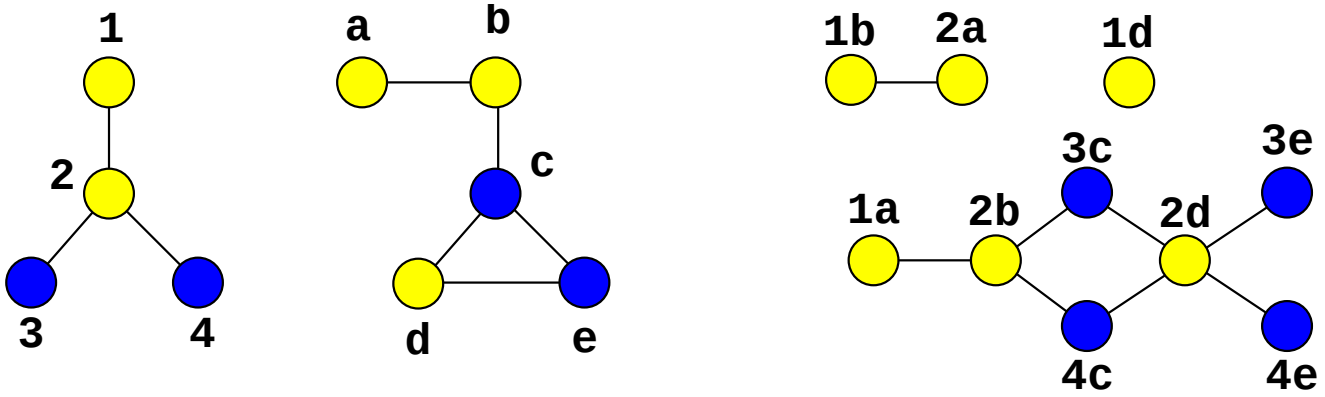
These three kernels (*n*th-order, random and geometric walk kernels) can be computed efficiently in polynomial time.

# 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. $E = \left\{\left((v_1, v_2), (v_1', v_2')\right) \in V \times V \ : \ (v_1, v_1') \in E_1 \text{ and } (v_2, v_2') \in E_2\right\}$.




**G1** **G2** **G1 x G2**

# 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

$$
\begin{aligned}
K_{walk}(G_1, G_2) &= \sum_{s \in \mathcal{S}} \Phi_s(G_1)\Phi_s(G_2) \\
&= \sum_{(w_1, w_2) \in \mathcal{W}(G_1) \times \mathcal{W}(G_1)} \lambda_{G_1}(w_1)\lambda_{G_2}(w_2)\mathbf{1}(l(w_1) = l(w_2)) \\
&= \sum_{w \in \mathcal{W}(G_1 \times G_2)} \lambda_{G_1 \times G_2}(w)\,.
\end{aligned}
$$

# 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

$$
\begin{aligned}
K_{walk}(G_1, G_2) &= \sum_{s \in \mathcal{S}} \Phi_s(G_1)\Phi_s(G_2) \\
&= \sum_{(w_1, w_2) \in \mathcal{W}(G_1) \times \mathcal{W}(G_1)} \lambda_{G_1}(w_1)\lambda_{G_2}(w_2)\mathbf{1}(l(w_1) = l(w_2)) \\
&= \sum_{w \in \mathcal{W}(G_1 \times G_2)} \lambda_{G_1 \times G_2}(w)\,.
\end{aligned}
$$

# 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:

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

- Computation in $O(n|G_1||G_2|d_1 d_2)$, 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 $w = v_1 \dots v_n$ can be decomposed as:

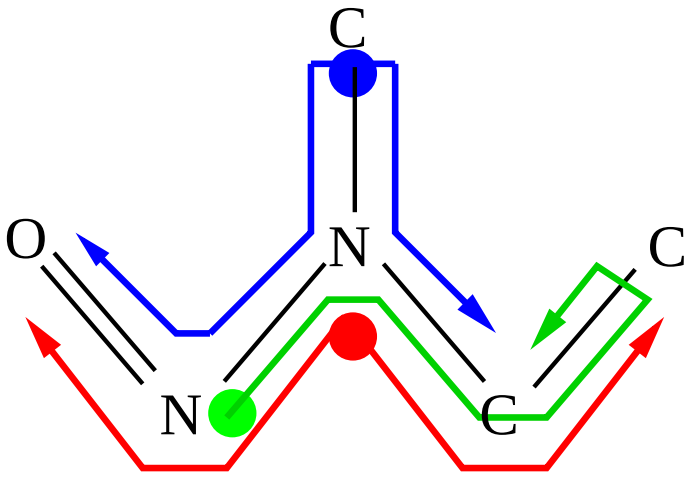
$$\lambda_G(v_1 \dots v_n) = \lambda^i(v_1) \prod_{i=2}^{n} \lambda^t(v_{i-1}, v_i)\,.$$

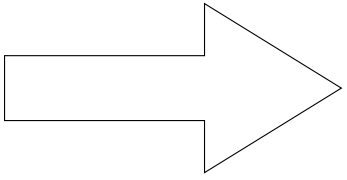
- Let $\Lambda_i$ be the vector of $\lambda^i(v)$ and $\Lambda_t$ be the matrix of $\lambda^t(v, v')$:

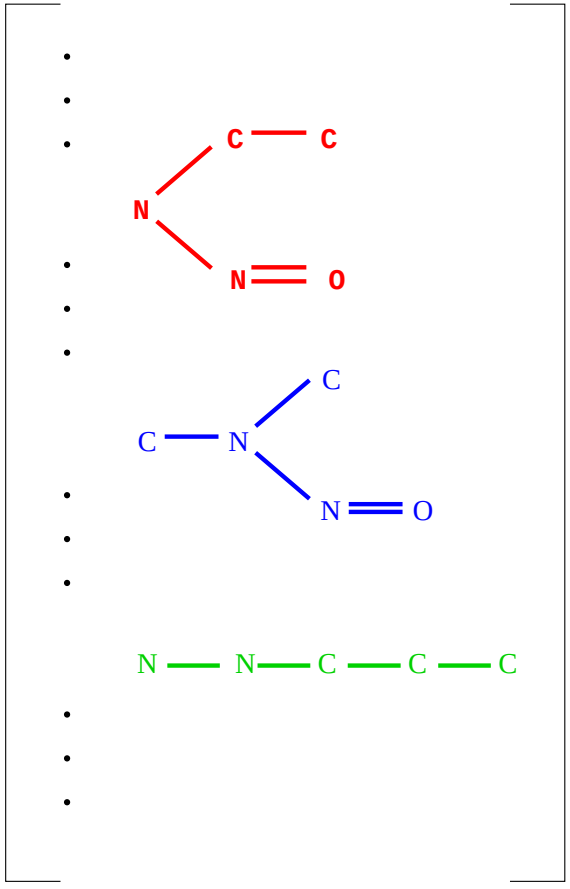
$$\begin{aligned} K_{walk}(G_1, G_2) &= \sum_{n=1}^{\infty} \sum_{w \in \mathcal{W}_n(G_1 \times G_2)} \lambda^i(v_1) \prod_{i=2}^{n} \lambda^t(v_{i-1}, v_i) \\ &= \sum_{n=0}^{\infty} \Lambda_i \Lambda_t^n \mathbf{1} \\ &= \Lambda_i \left(I - \Lambda_t\right)^{-1} \mathbf{1} \end{aligned}$$

- Computation in $O(|G_1|^3 |G_2|^3)$

# Extension: branching walks (Ramon and Gärtner, 2003; Mahé and Vert, 2009)

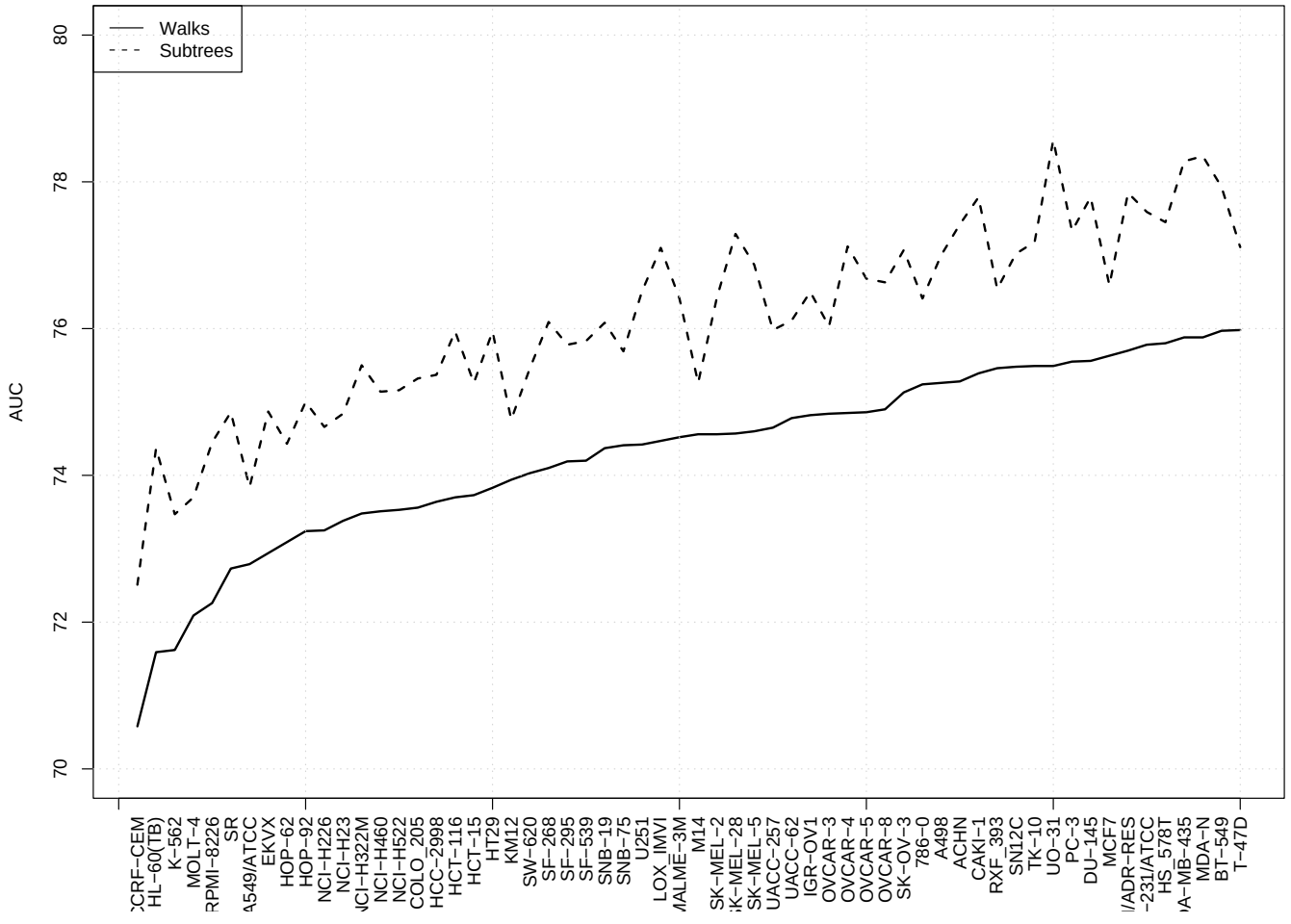





$$\mathcal{T}(v, n+1) = \sum_{R \subset \mathcal{N}(v)} \prod_{v' \in R} \lambda_t(v, v') \mathcal{T}(v', n)\,,$$

# 2D Subtree vs walk kernels

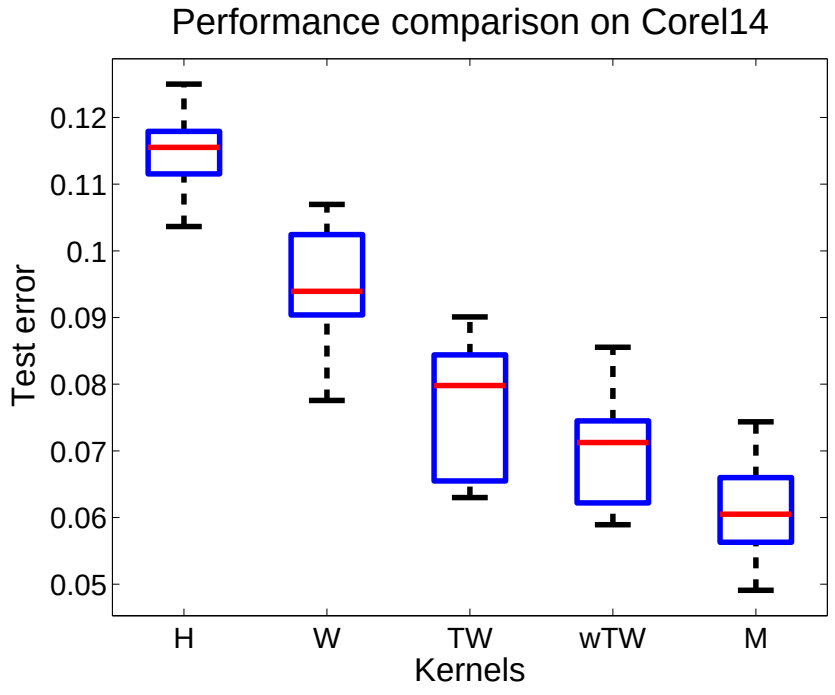



Screening of inhibitors for 60 cancer cell lines.

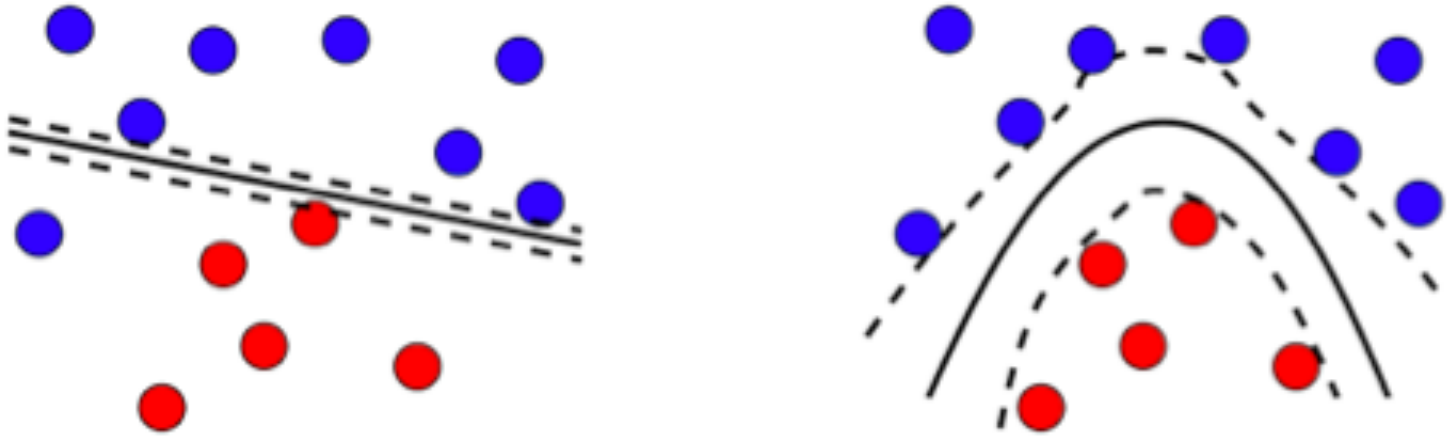
# Image classification (Harchaoui and Bach, 2007)

## 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).



# SVM summary



- Large margin classifier
- Control of the regularization / data fitting trade-off with $C$
- Linear or nonlinear (with the kernel trick)
- Extension to strings, graphs... and many other

# References


N. Aronszajn. Theory of reproducing kernels. *Trans. Am. Math. Soc.*, 68:337 – 404, 1950. URL http://www.jstor.org/stable/1990404.

K. M. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, pages 74–81, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2278-5. doi: http://dx.doi.org/10.1109/ICDM.2005.132.

Z. Harchaoui and F. Bach. Image classification with segmentation graph kernels. In *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, pages 1–8. IEEE Computer Society, 2007. doi: 10.1109/CVPR.2007.383049. URL http://dx.doi.org/10.1109/CVPR.2007.383049.

D. Haussler. Convolution Kernels on Discrete Structures. Technical Report UCSC-CRL-99-10, UC Santa Cruz, 1999.

C. Helma, T. Cramer, S. Kramer, and L. De Raedt. Data mining and machine learning techniques for the identification of mutagenicity inducing substructures and structure activity relationships of noncongeneric compounds. *J. Chem. Inf. Comput. Sci.*, 44(4):1402–11, 2004. doi: 10.1021/ci034254q. URL http://dx.doi.org/10.1021/ci034254q.

C. Leslie and R. Kuang. Fast string kernels using inexact matching for protein sequences. *J. Mach. Learn. Res.*, 5:1435–1455, 2004.

C. Leslie, E. Eskin, and W. Noble. The spectrum kernel: a string kernel for SVM protein classification. In R. B. Altman, A. K. Dunker, L. Hunter, K. Lauerdale, and T. E. Klein, editors, *Proceedings of the Pacific Symposium on Biocomputing 2002*, pages 564–575, Singapore, 2002. World Scientific.

# References (cont.)


H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. n. p. v. d. d. r. Watkins. Text classification using string kernels. *J. Mach. Learn. Res.*, 2:419–444, 2002. URL `http://www.ai.mit.edu/projects/jmlr/papers/volume2/lodhi02a/abstract.html`.

P. Mahé and J. P. Vert. Graph kernels based on tree patterns for molecules. *Mach. Learn.*, 75(1): 3–35, 2009. doi: 10.1007/s10994-008-5086-2. URL `http://dx.doi.org/10.1007/s10994-008-5086-2`.

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

J. Ramon and T. Gärtner. Expressivity versus efficiency of graph kernels. In T. Washio and L. De Raedt, editors, *Proceedings of the First International Workshop on Mining Graphs, Trees and Sequences*, pages 65–74, 2003.

H. Saigo, J.-P. Vert, N. Ueda, and T. Akutsu. Protein homology detection using string alignment kernels. *Bioinformatics*, 20(11):1682–1689, 2004. URL `http://bioinformatics.oupjournals.org/cgi/content/abstract/20/11/1682`.

N. Sherashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt. Efficient graphlet kernels for large graph comparison. In *12th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 488–495, Clearwater Beach, Florida USA, 2009. Society for Artificial Intelligence and Statistics.