Support vector machines, kernels, and applications in computational biology

Jean-Philippe Vert
Jean-Philippe.Vert@mines-paristech.fr

Mines ParisTech / Institut Curie / Inserm

Mines ParisTech, ES "Machine learning" module.

Outline

1. Machine learning in bioinformatics
2. Linear support vector machines
3. Nonlinear SVM and kernels
4. SVM for complex data: the case of graphs
5. Conclusion

Outline

1. Machine learning in bioinformatics
2. Linear support vector machines
3. Nonlinear SVM and kernels
4. SVM for complex data: the case of graphs
5. Conclusion

Outline

1. Machine learning in bioinformatics
2. Linear support vector machines
3. Nonlinear SVM and kernels
4. SVM for complex data: the case of graphs
5. Conclusion

Outline

1. Machine learning in bioinformatics
2. Linear support vector machines
3. Nonlinear SVM and kernels
4. SVM for complex data: the case of graphs
5. Conclusion

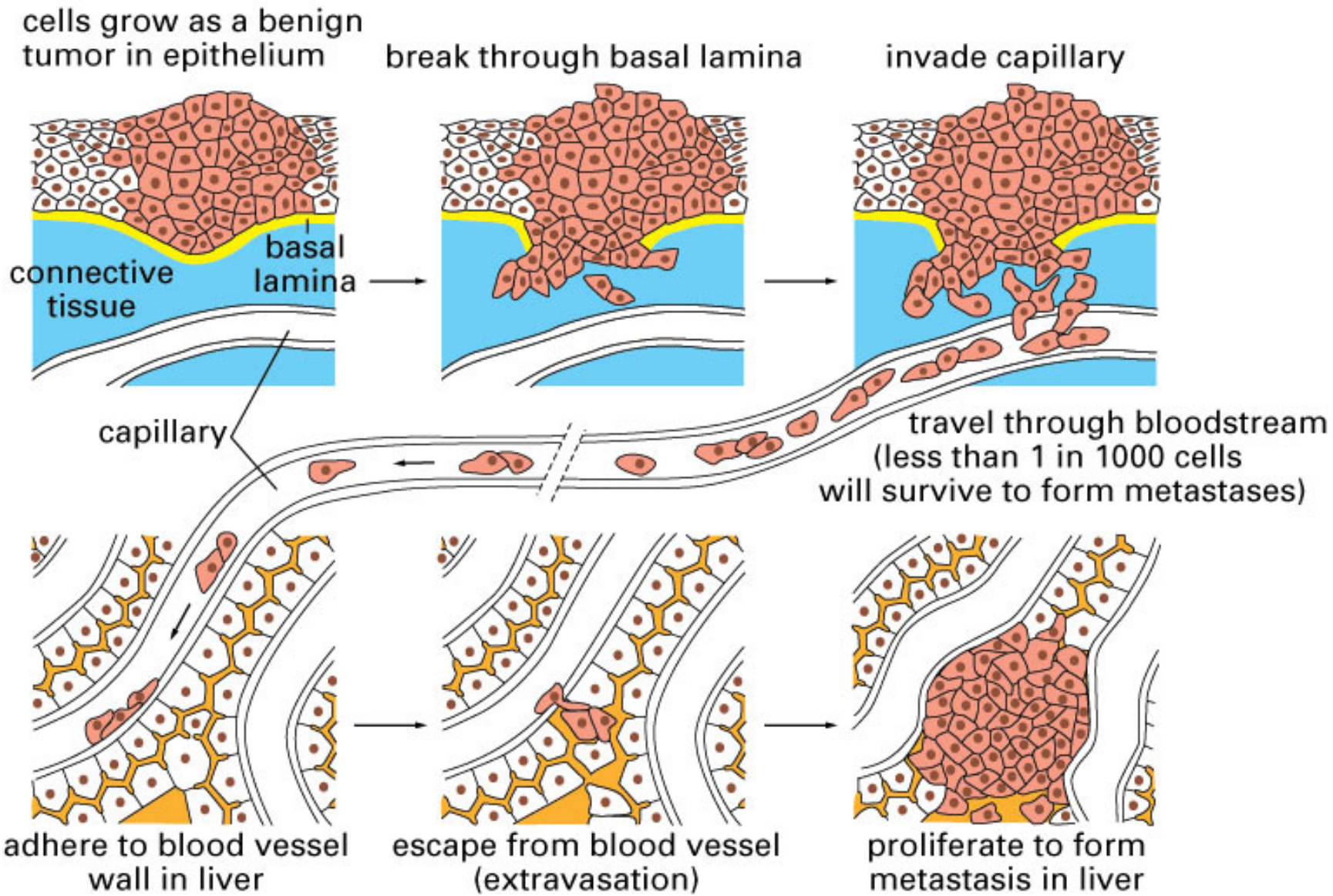
Outline

1. Machine learning in bioinformatics
2. Linear support vector machines
3. Nonlinear SVM and kernels
4. SVM for complex data: the case of graphs
5. Conclusion

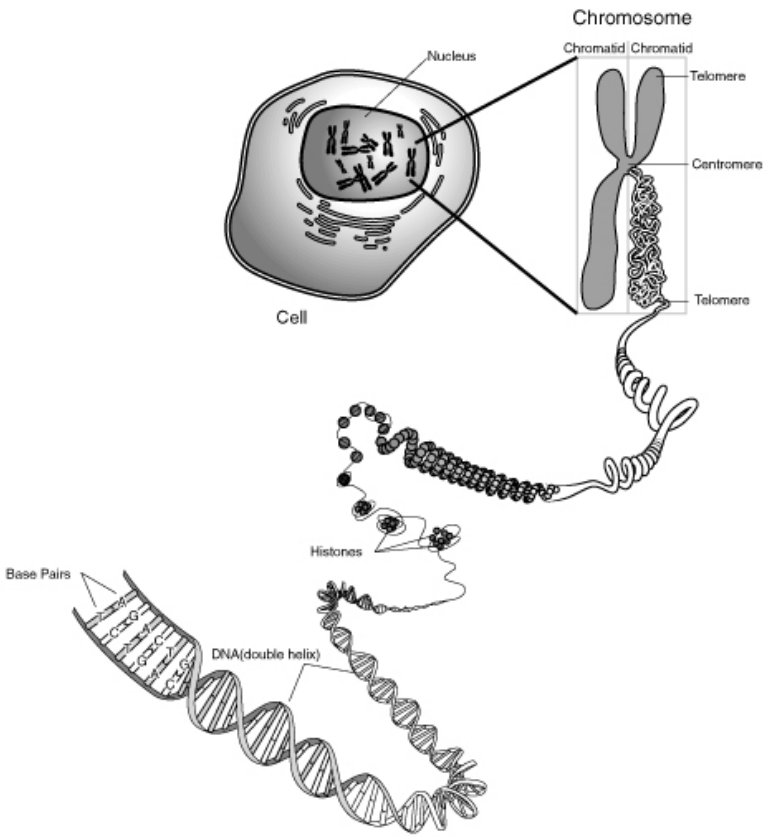
Outline

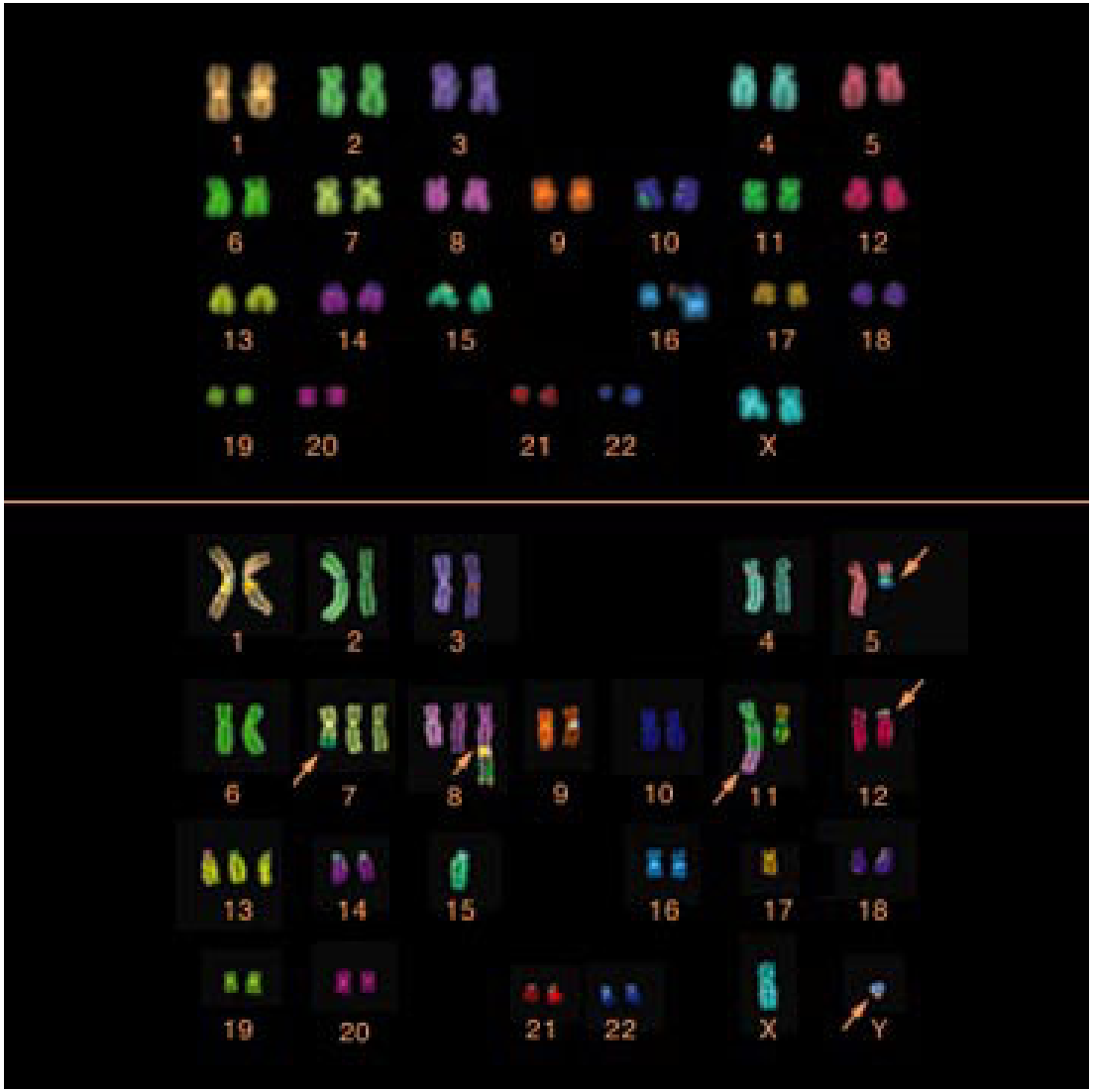
1. Machine learning in bioinformatics
2. Linear support vector machines
3. Nonlinear SVM and kernels
4. SVM for complex data: the case of graphs
5. Conclusion

A simple view of cancer progression



Chromosomic aberrations in cancer

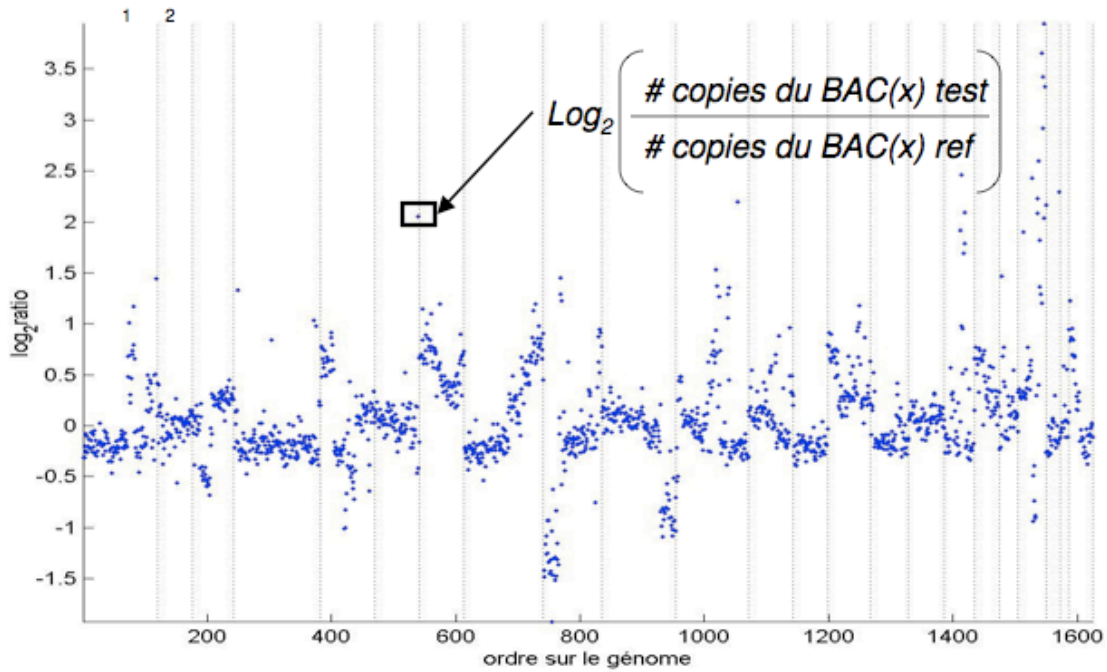




Comparative Genomic Hybridization (CGH)

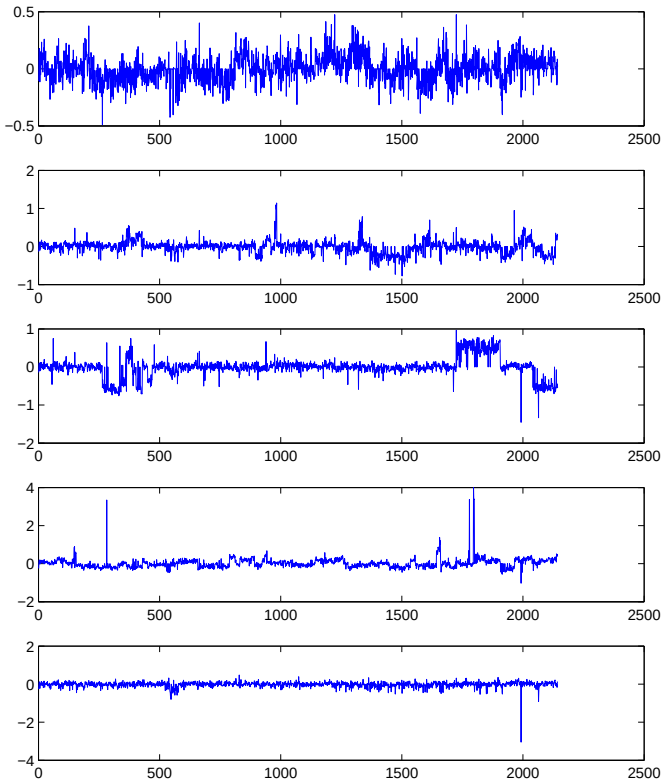
Motivation

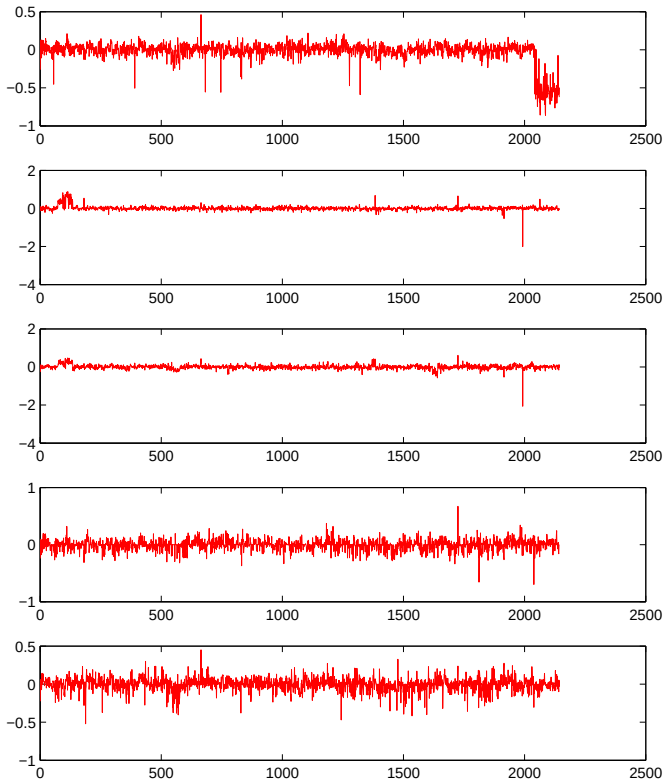
- Comparative genomic hybridization (CGH) data measure the DNA copy number along the genome
- Very useful, in particular in cancer research
- Can we classify CGH arrays for diagnosis or prognosis purpose?



Jain et al. Genome research 2002 12:325-332

Aggressive vs non-aggressive melanoma

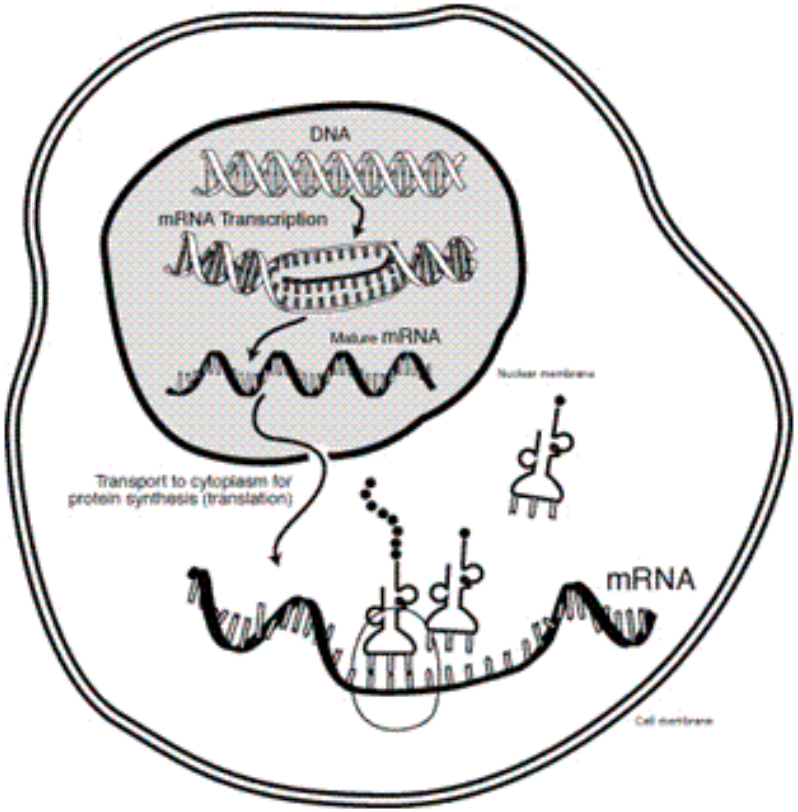


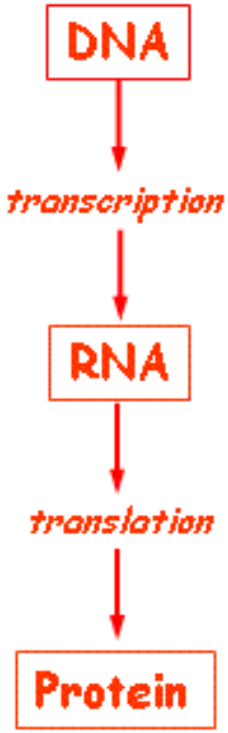


Problem 1

Given the CGH profile of a melanoma, is it aggressive or not?

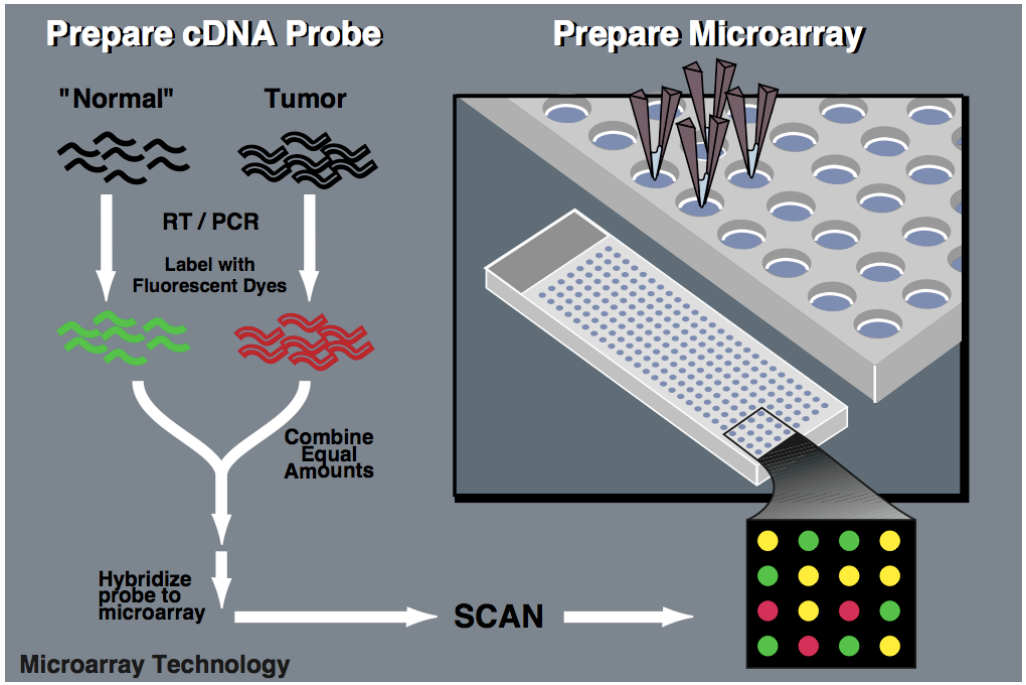
DNA → RNA → protein

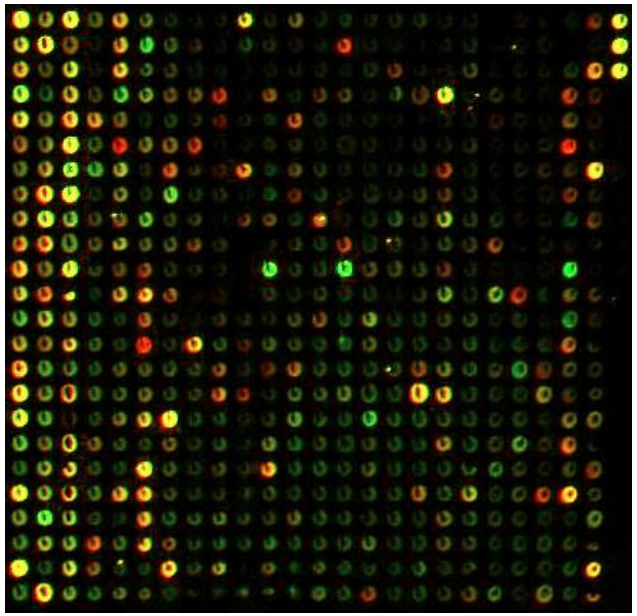




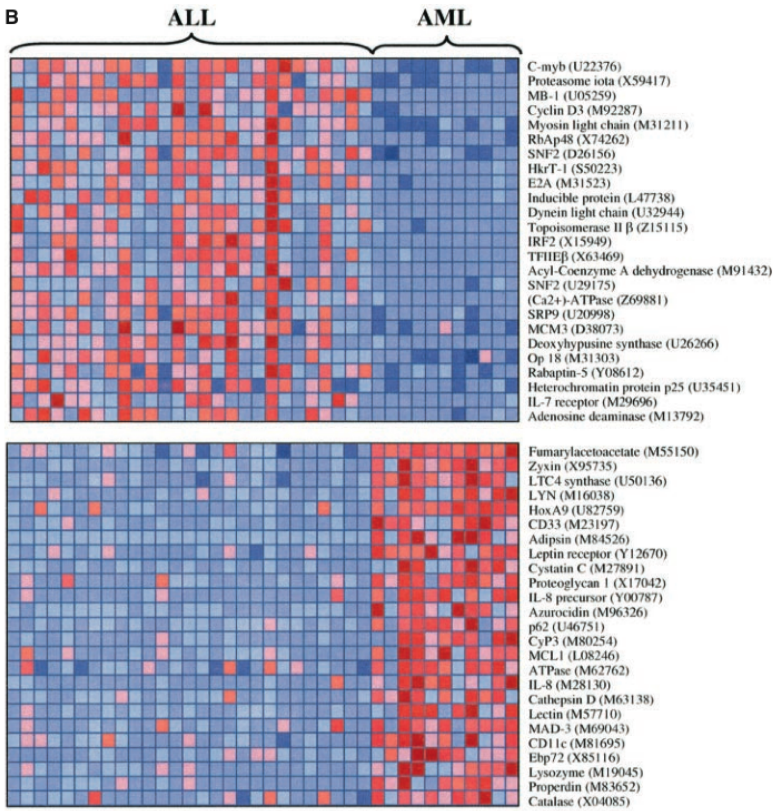
- CGH shows the (static) DNA
- Cancer cells have also abnormal (dynamic) gene expression (= transcription)

Tissue profiling with DNA chips





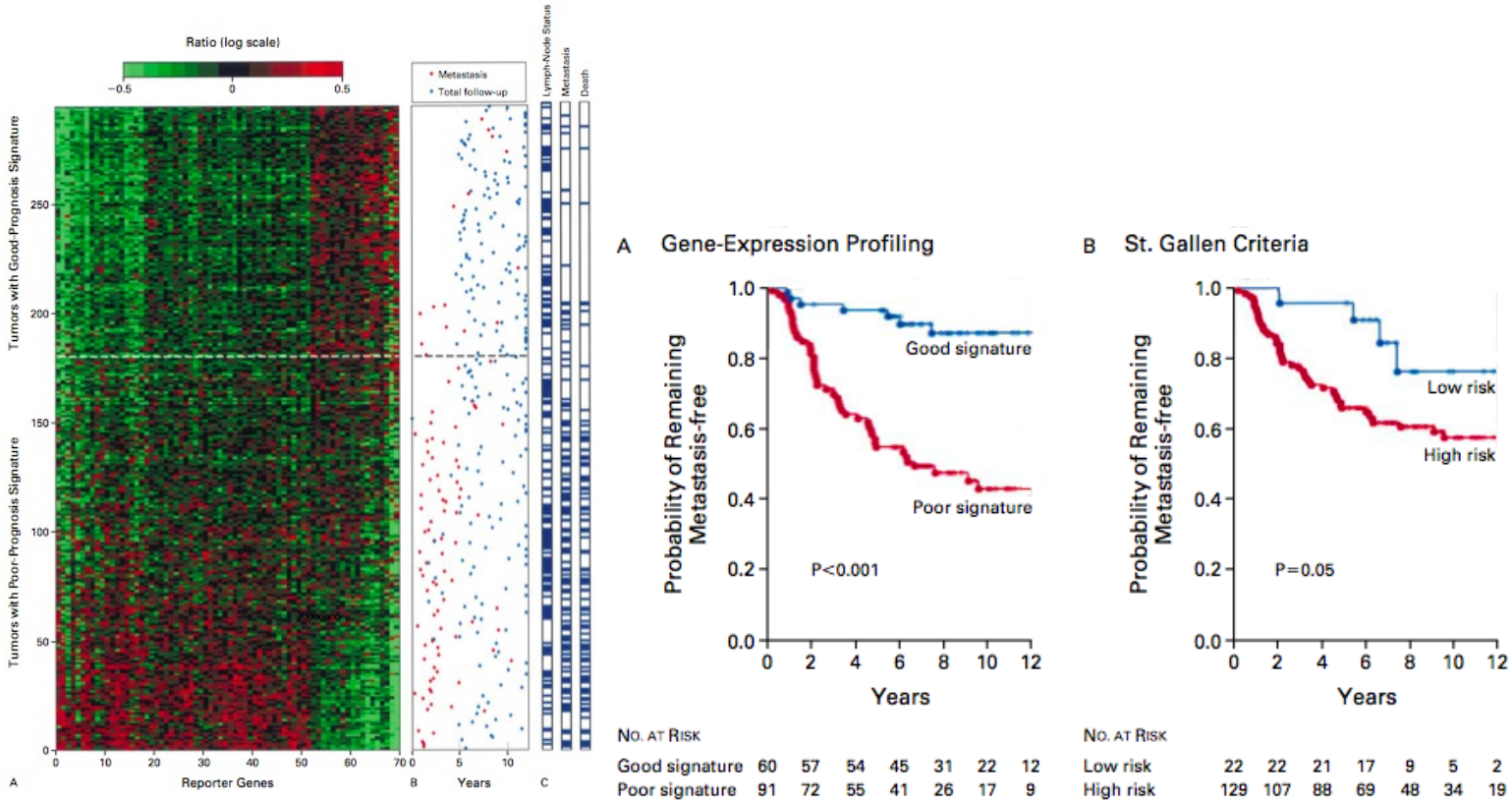
Use in diagnosis



Problem 2

Given the expression profile of a leukemia, is it an acute lymphocytic or myeloid leukemia (ALL or AML)?

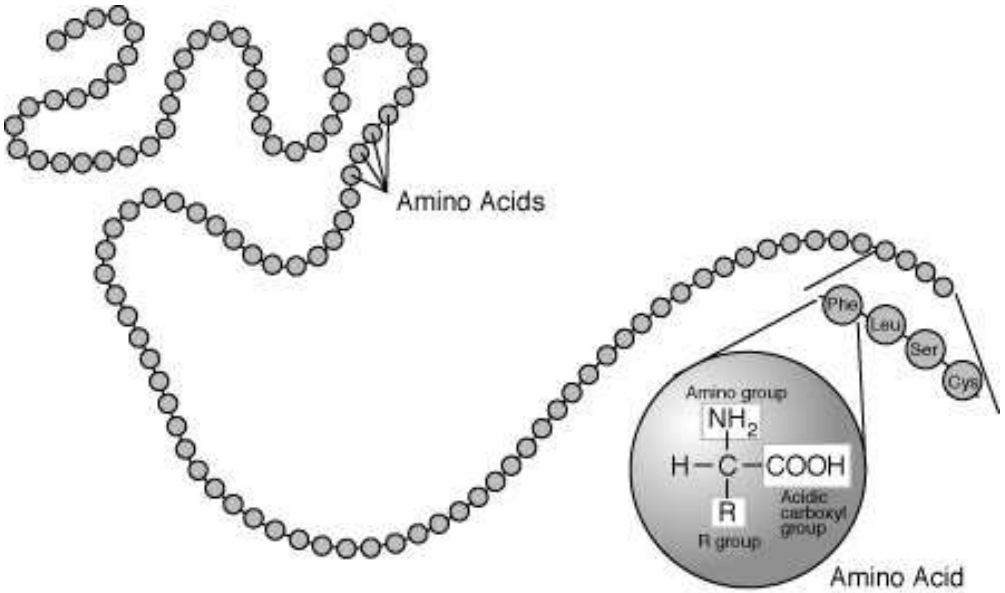
Use in prognosis



Problem 3

Given the expression profile of a breast cancer, is the risk of relapse within 5 years high?

Proteins



A : Alanine	V : Valine	L : Leucine
F : Phenylalanine	P : Proline	M : Methionine
E : Acide glutamique	K : Lysine	R : Arginine
T : Threonine	C : Cysteine	N : Asparagine
H : Histidine	V : Thyrosine	W : Tryptophane
I : Isoleucine	S : Serine	Q : Glutamine
D : Acide aspartique	G : Glycine	

Protein annotation

Data available

- Secreted proteins:

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

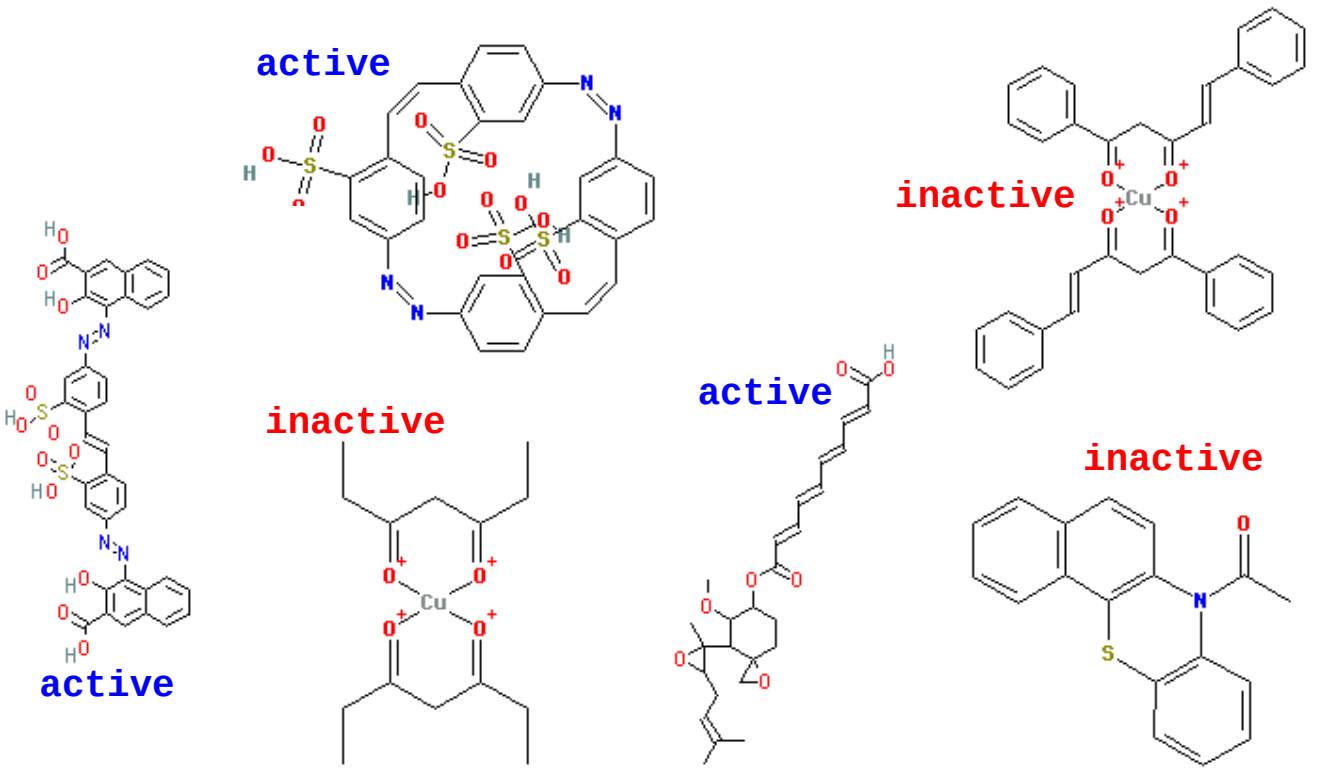
- Non-secreted proteins:

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

Problem 4

Given a newly sequenced protein, is it secreted or not?

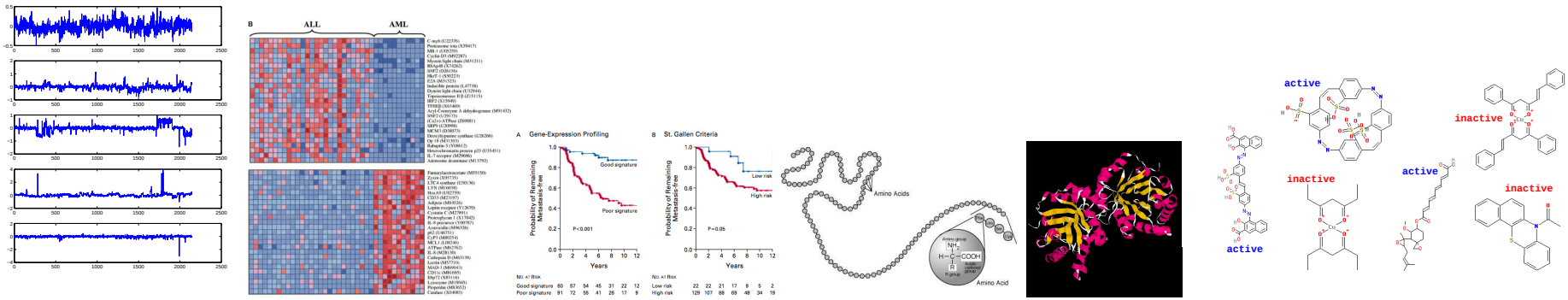
Drug discovery

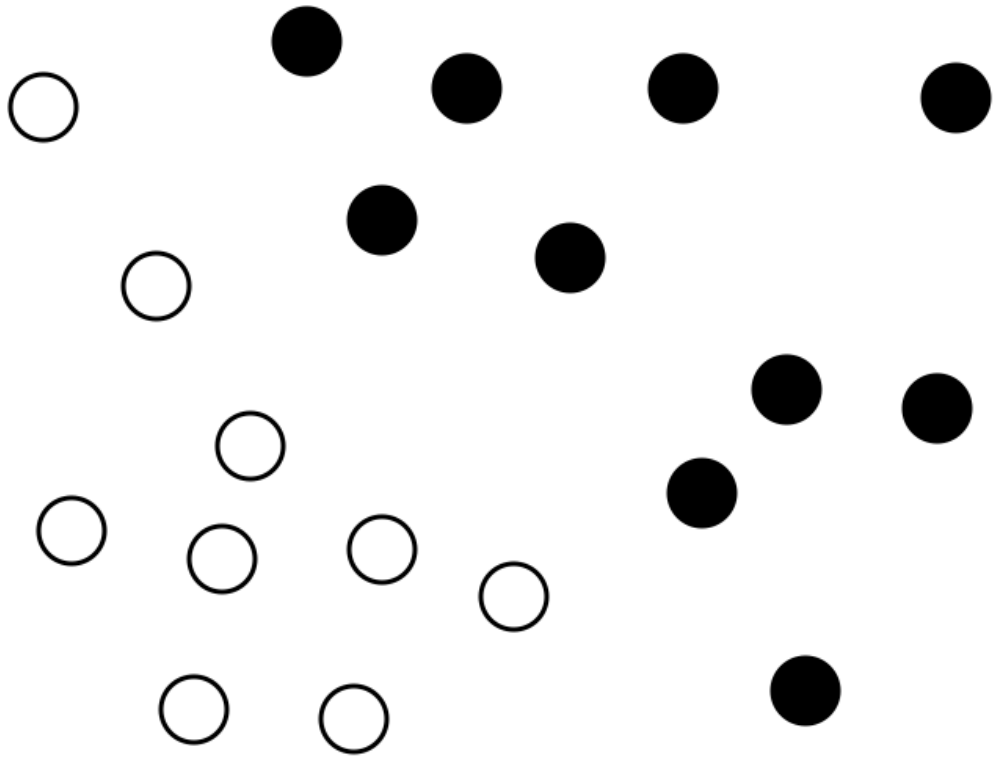


Problem 4

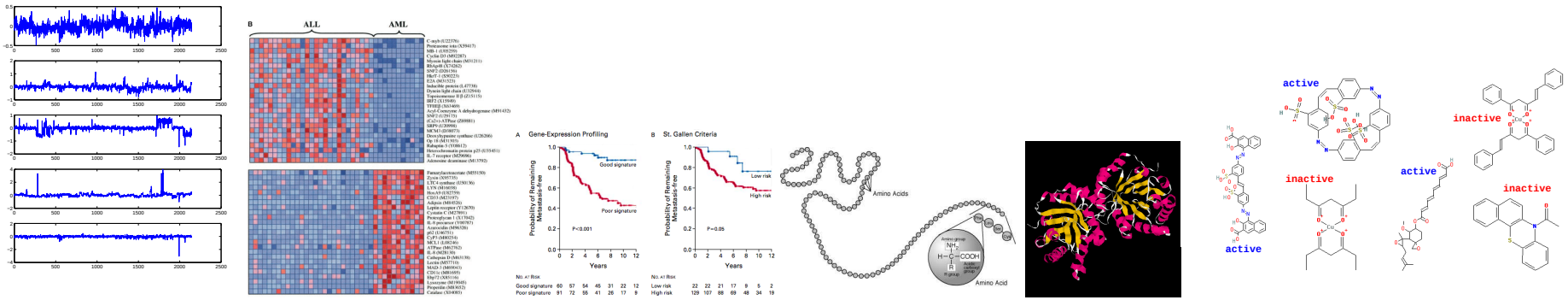
Given a new candidate molecule, is it likely to be active?

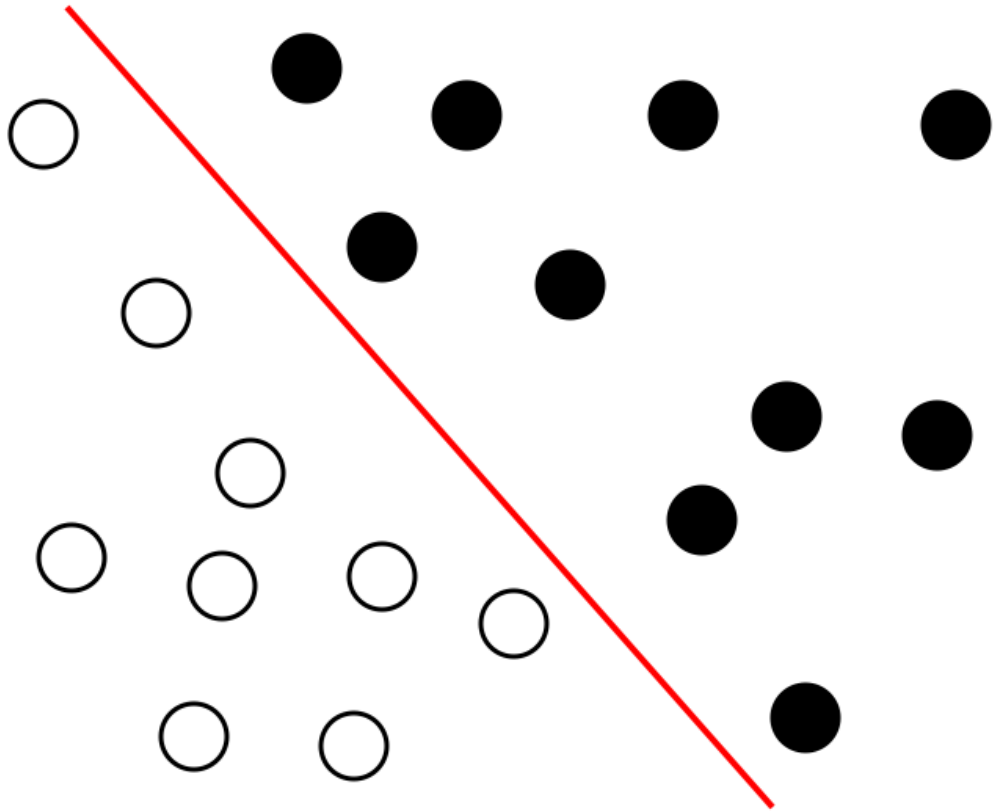
Pattern recognition, *aka* supervised classification



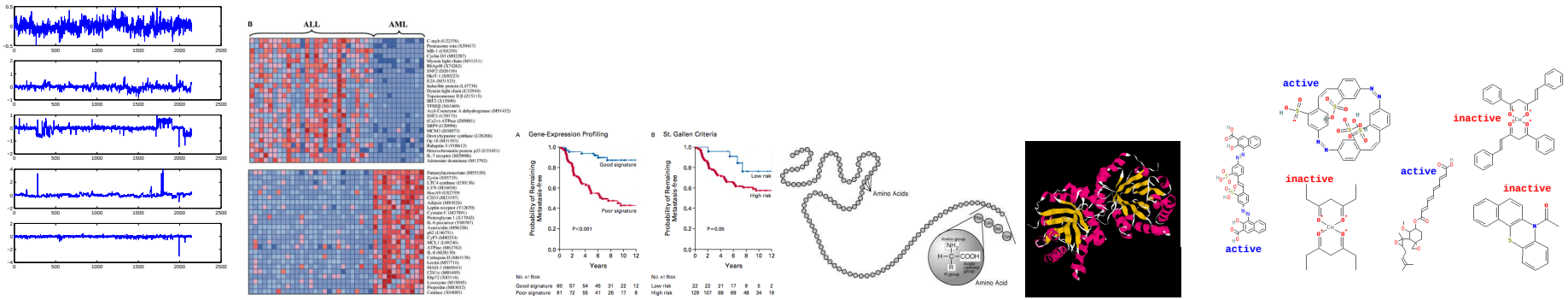


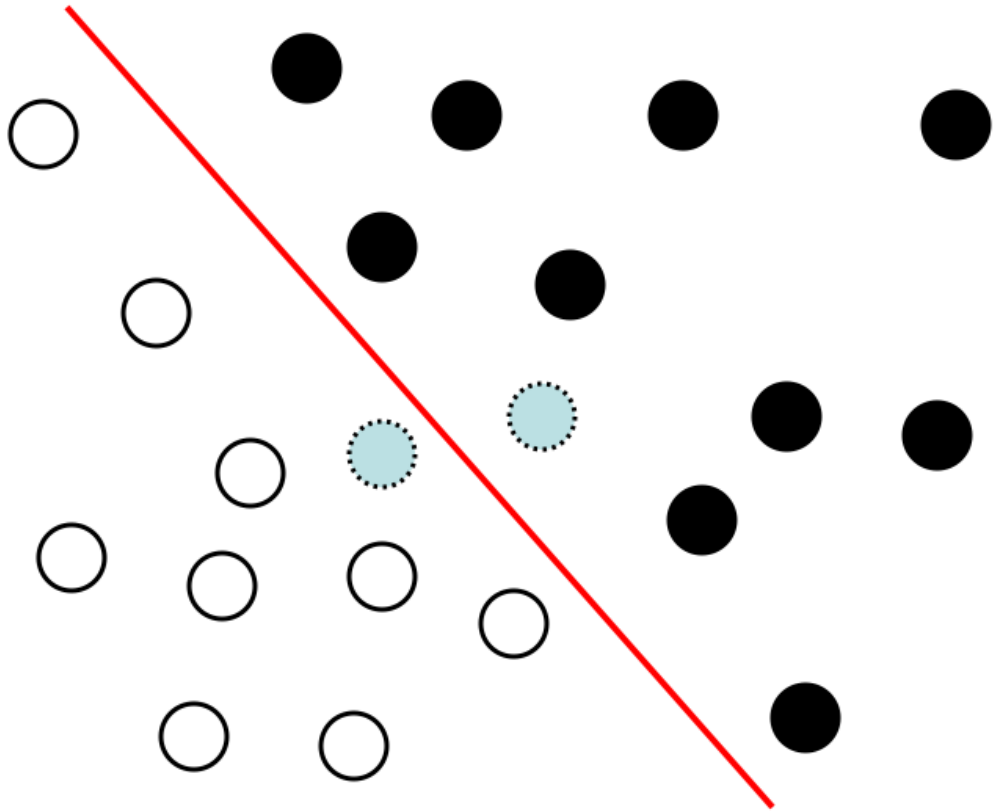
Pattern recognition, *aka* supervised classification



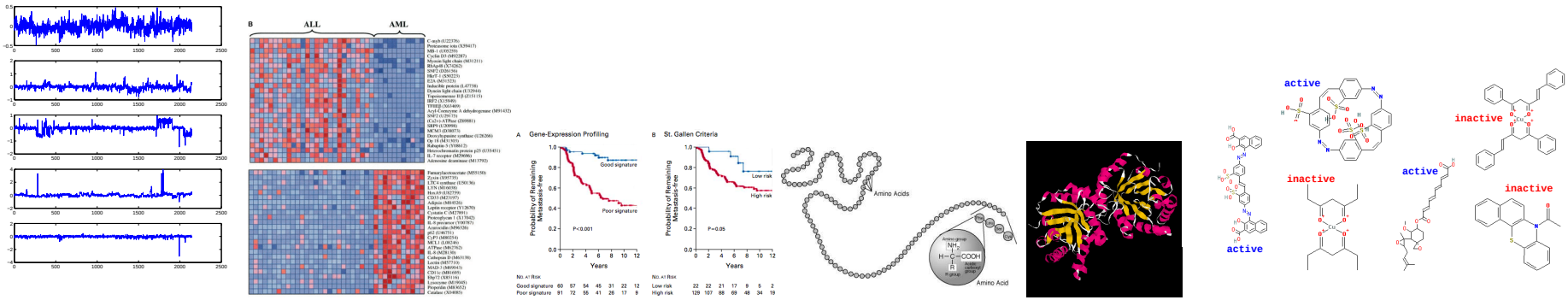


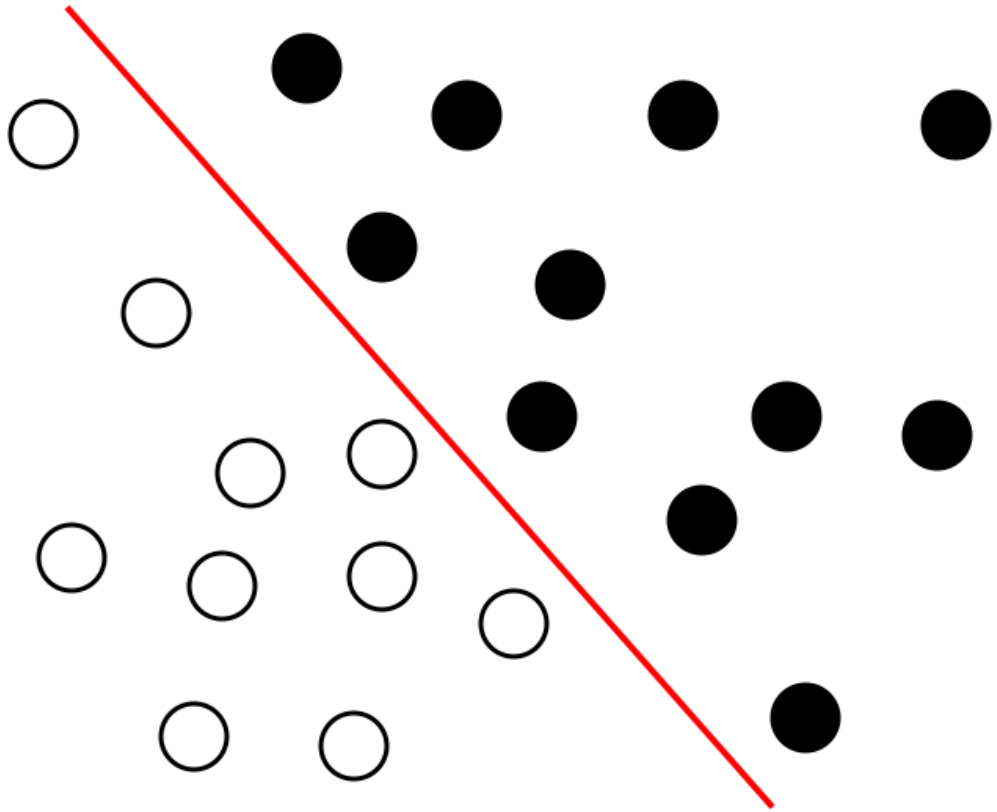
Pattern recognition, *aka* supervised classification



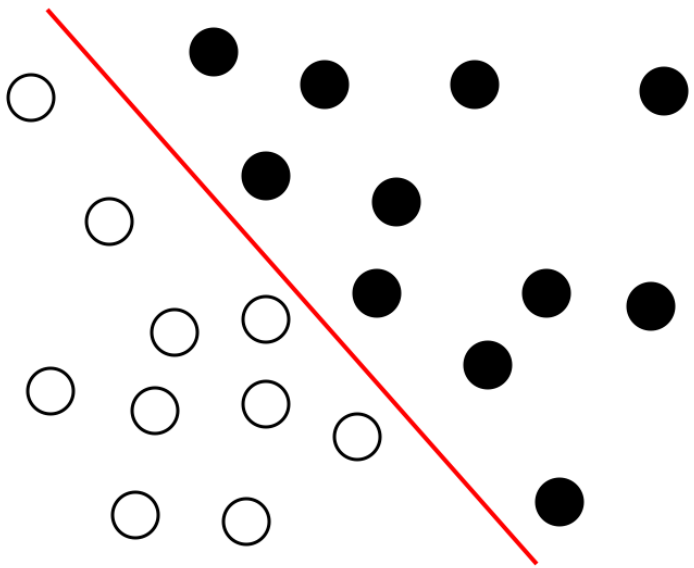


Pattern recognition, *aka* supervised classification





Pattern recognition, *aka* supervised classification



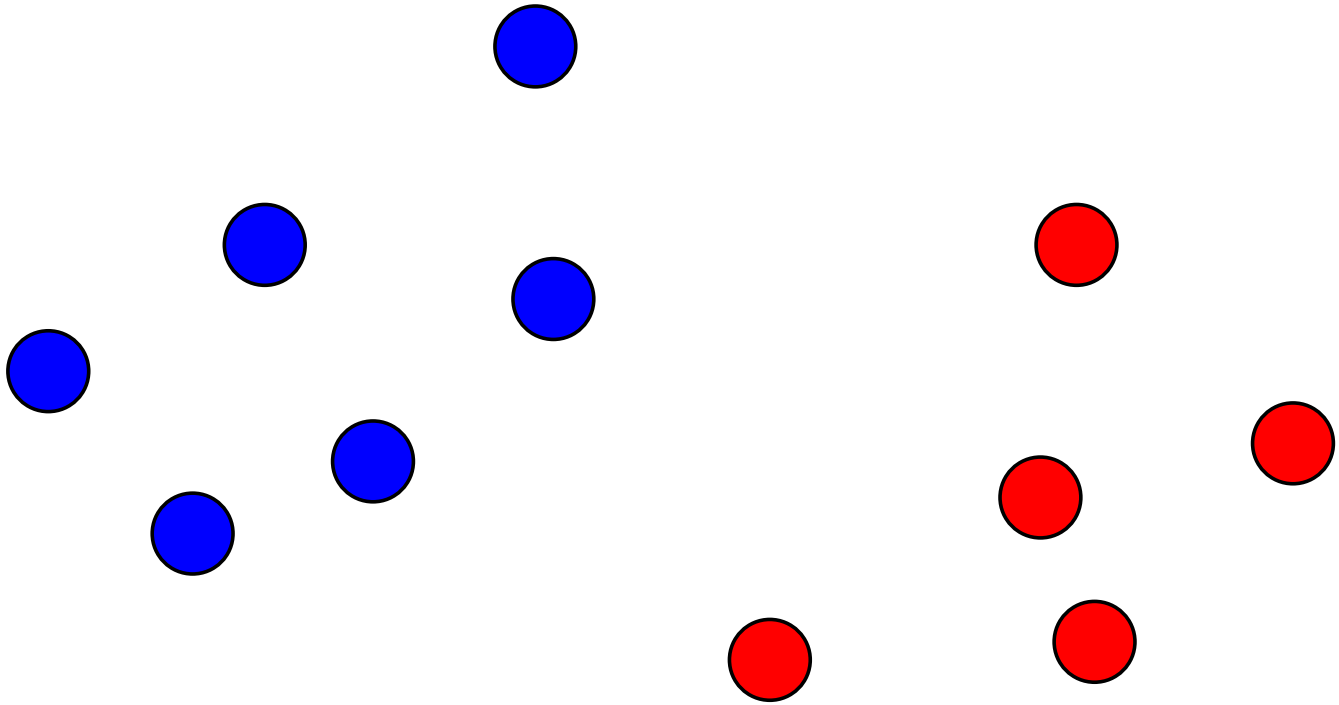
Challenges

- High dimension
- Few samples
- Structured data
- Heterogeneous data
- Prior knowledge
- Fast and scalable implementations
- Interpretable models

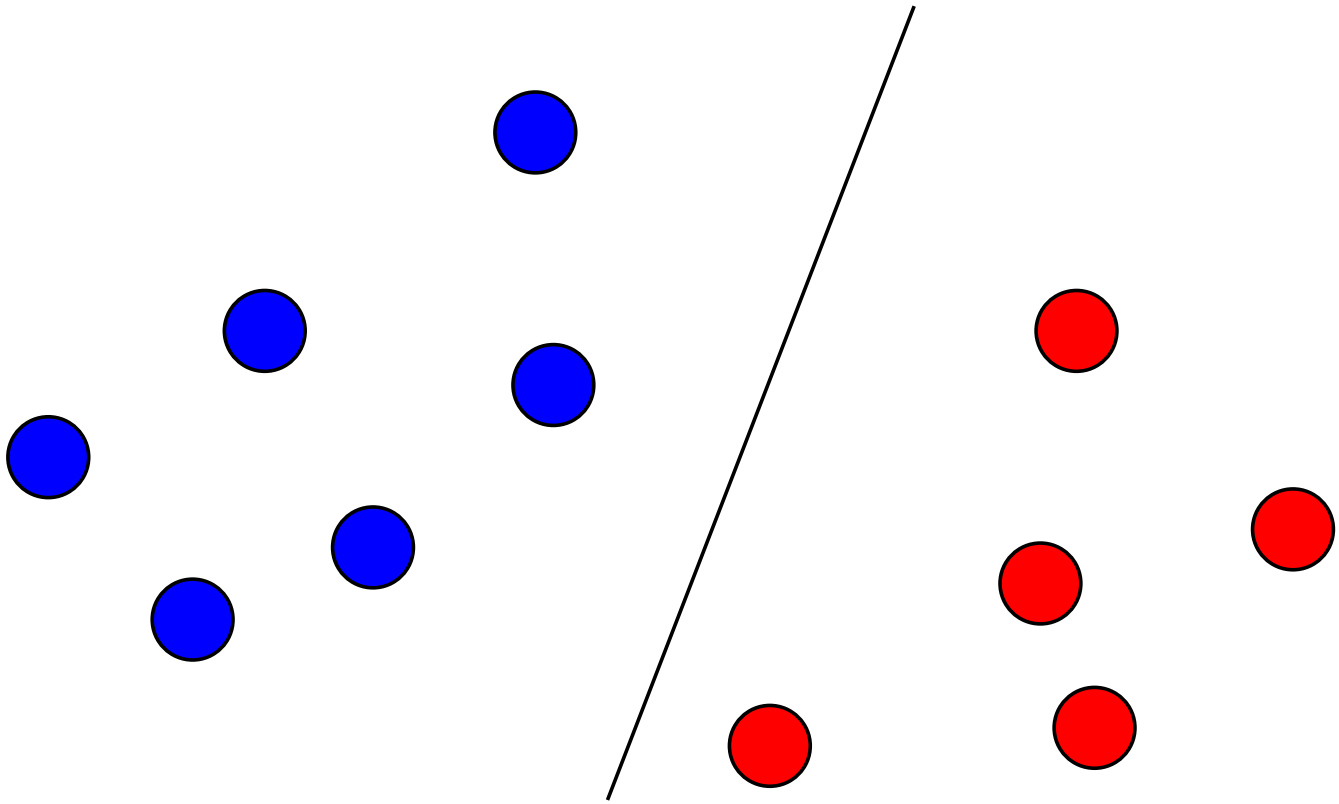
Outline

1. Machine learning in bioinformatics
2. Linear support vector machines
3. Nonlinear SVM and kernels
4. SVM for complex data: the case of graphs
5. Conclusion

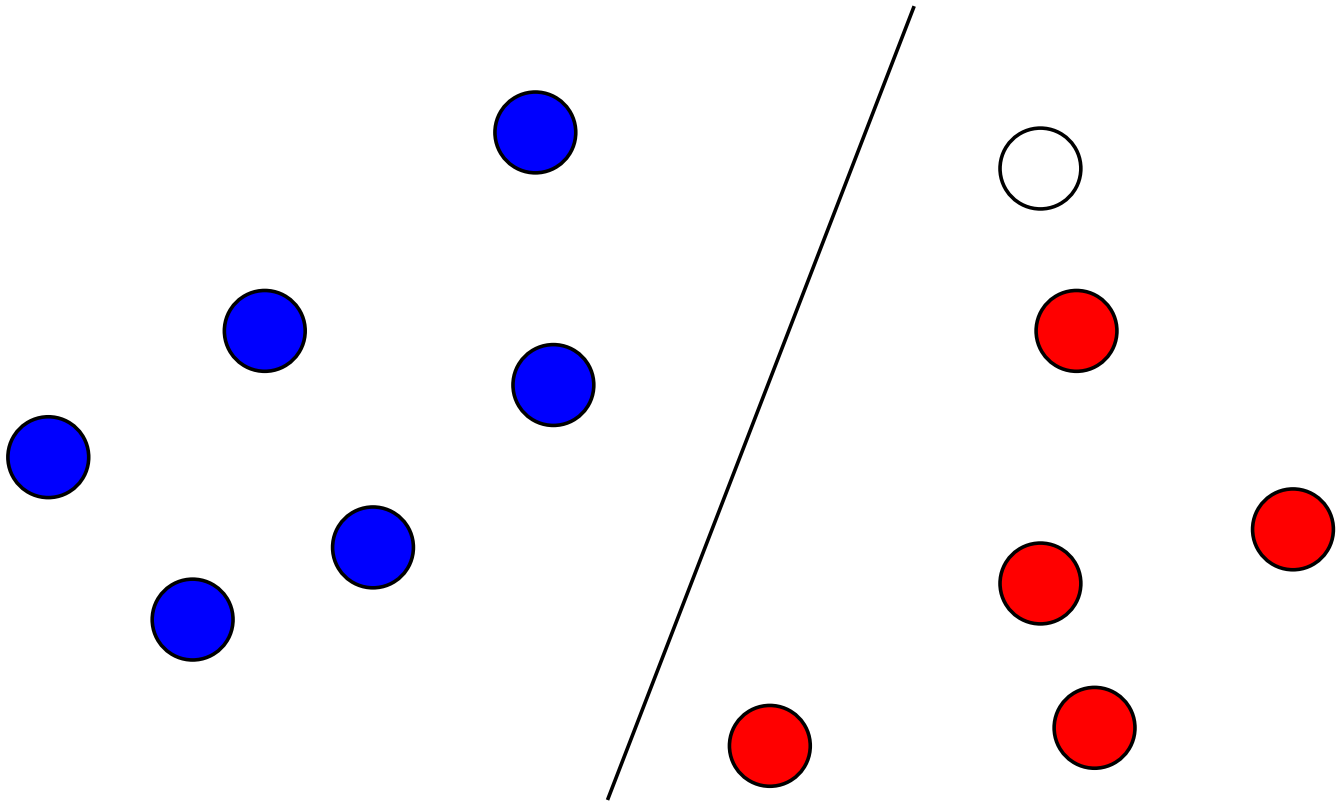
Linear classifiers



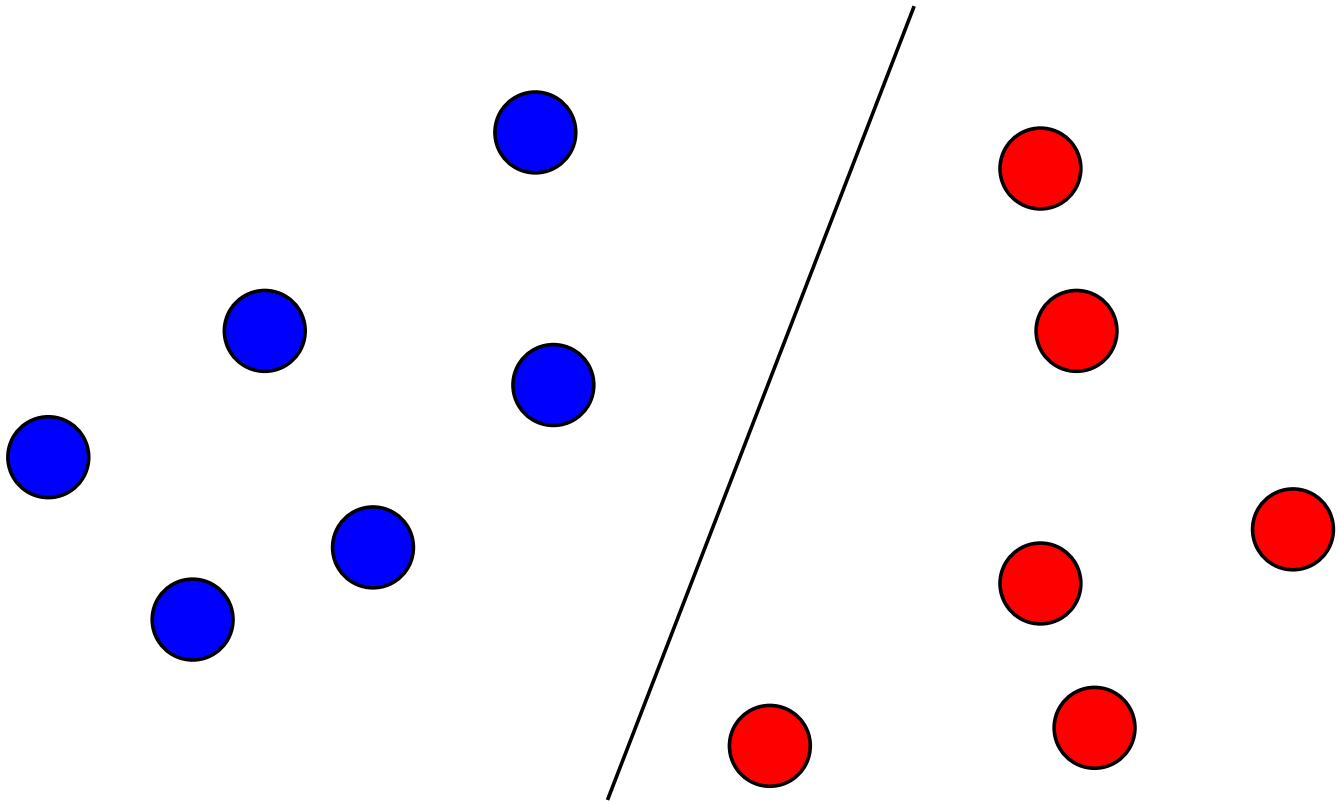
Linear classifiers



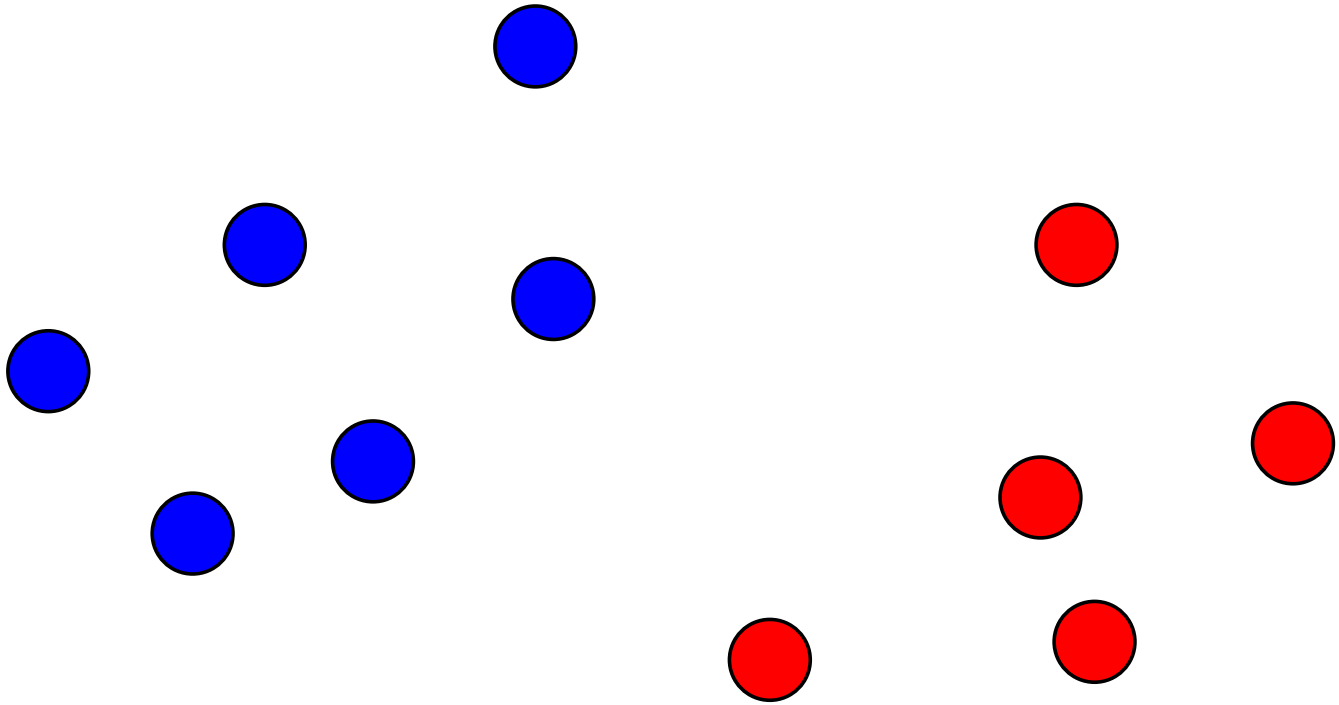
Linear classifiers



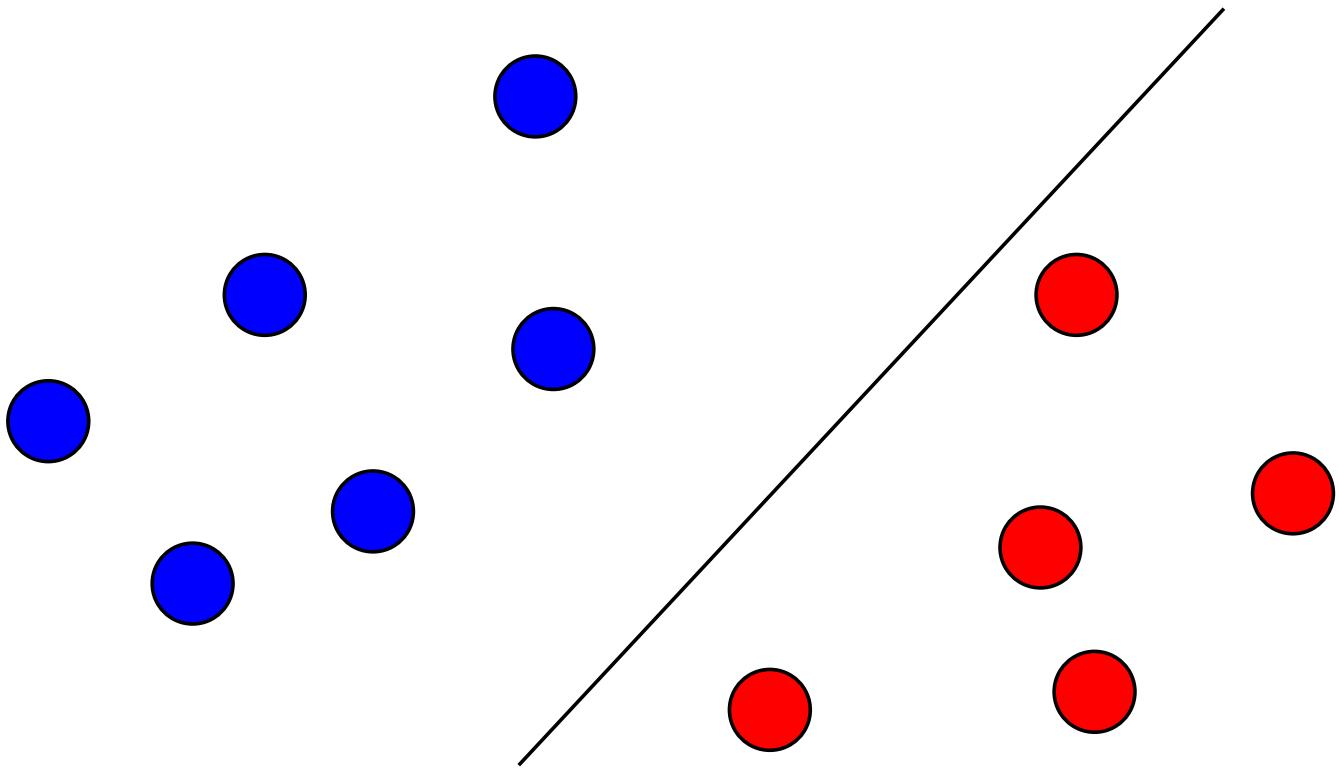
Linear classifiers



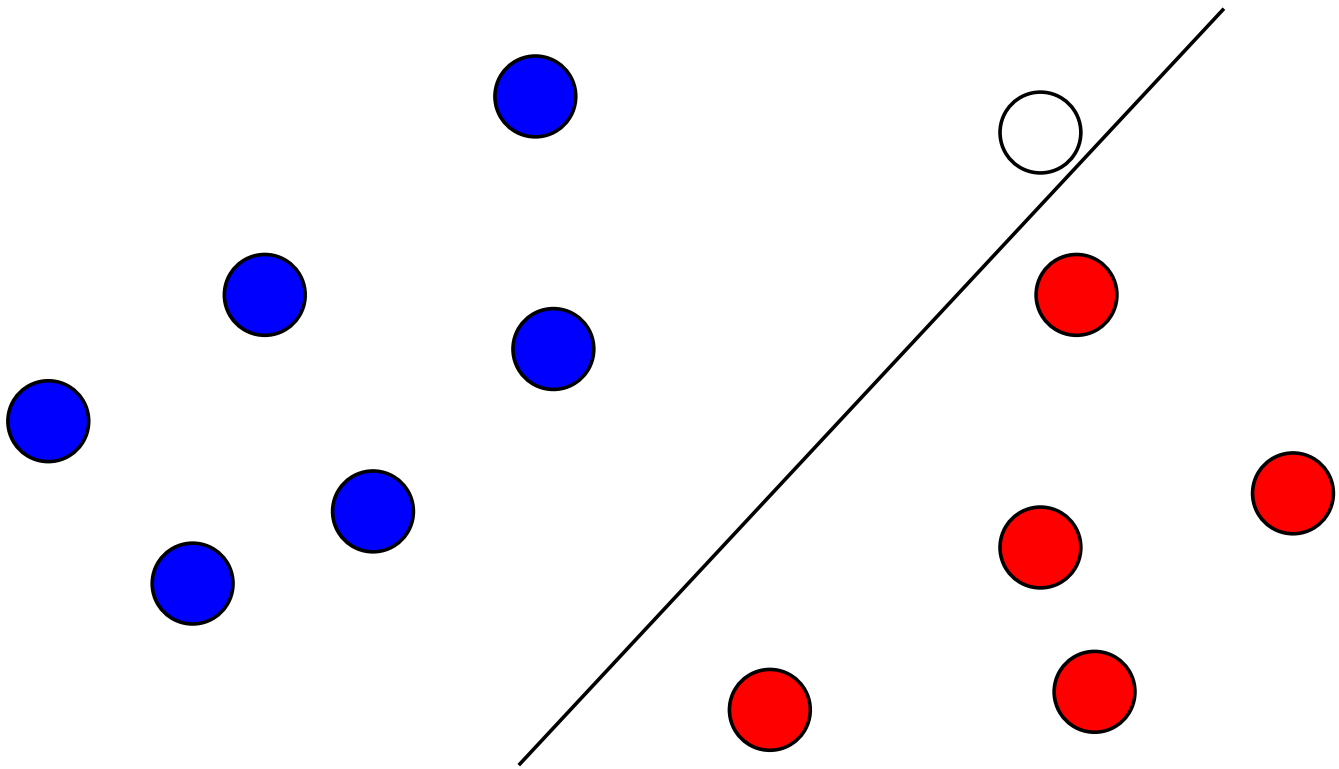
Linear classifiers



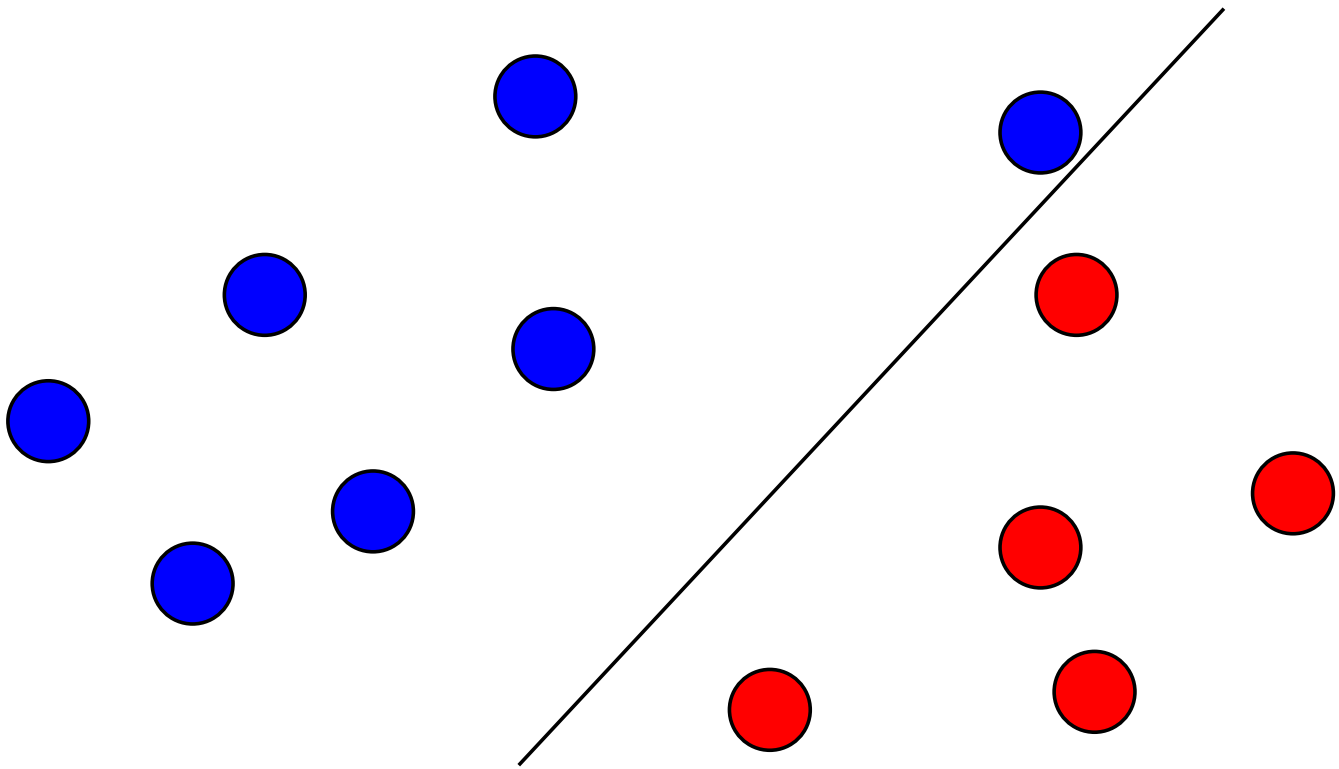
Linear classifiers



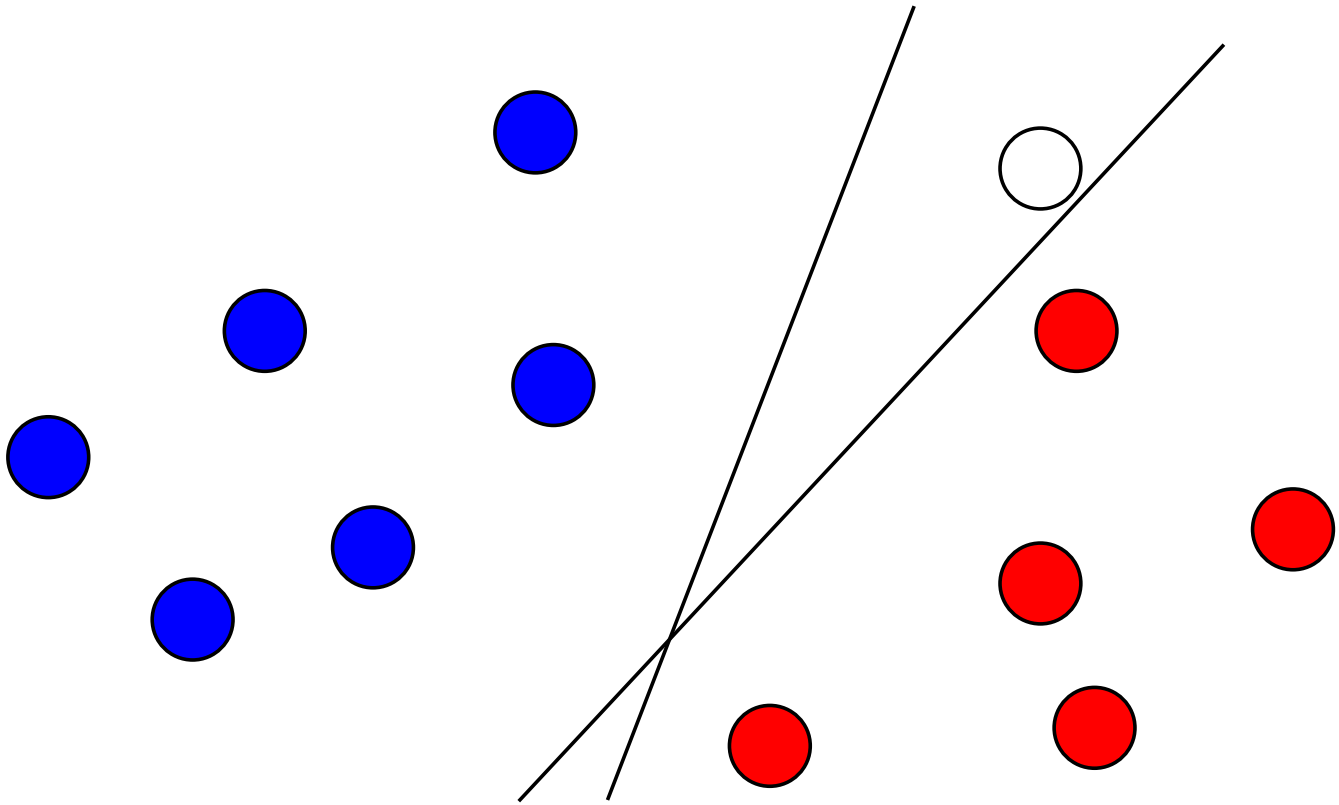
Linear classifiers



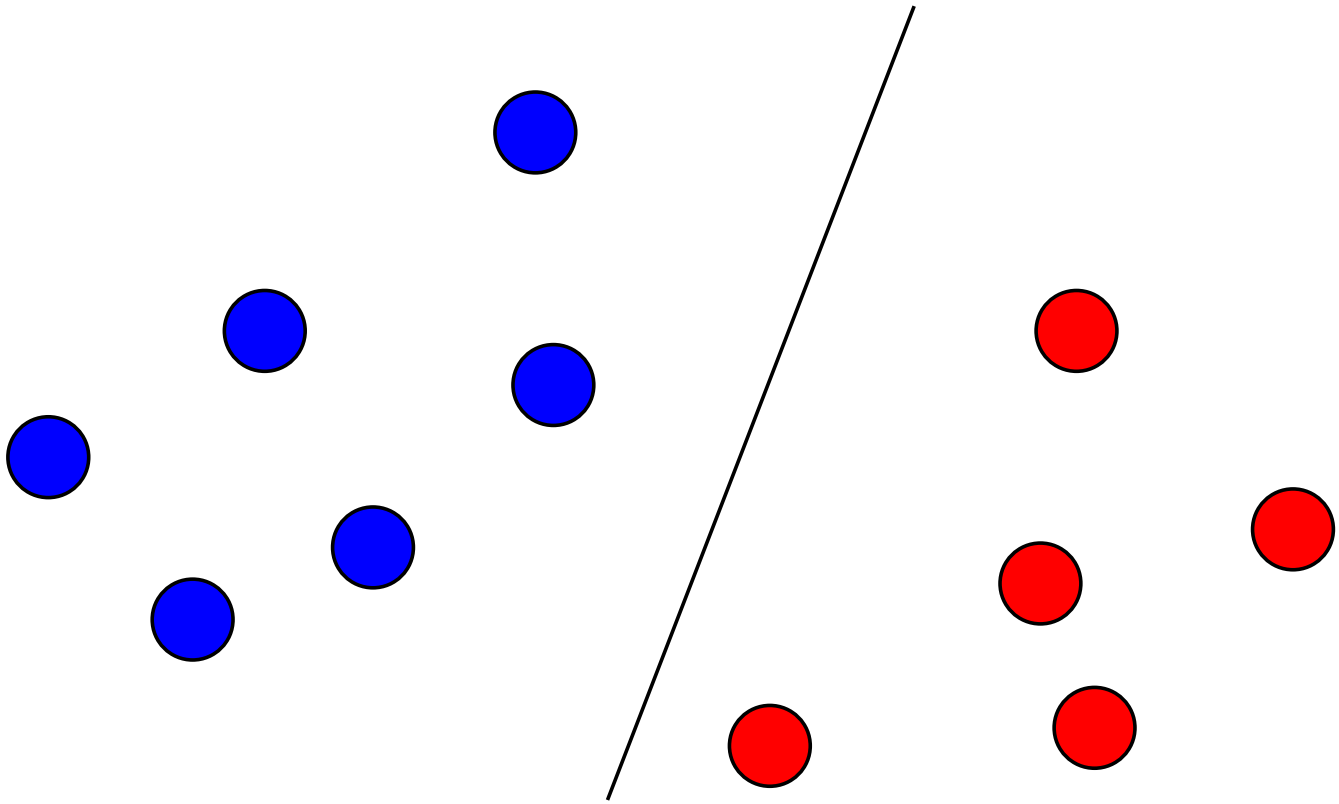
Linear classifiers



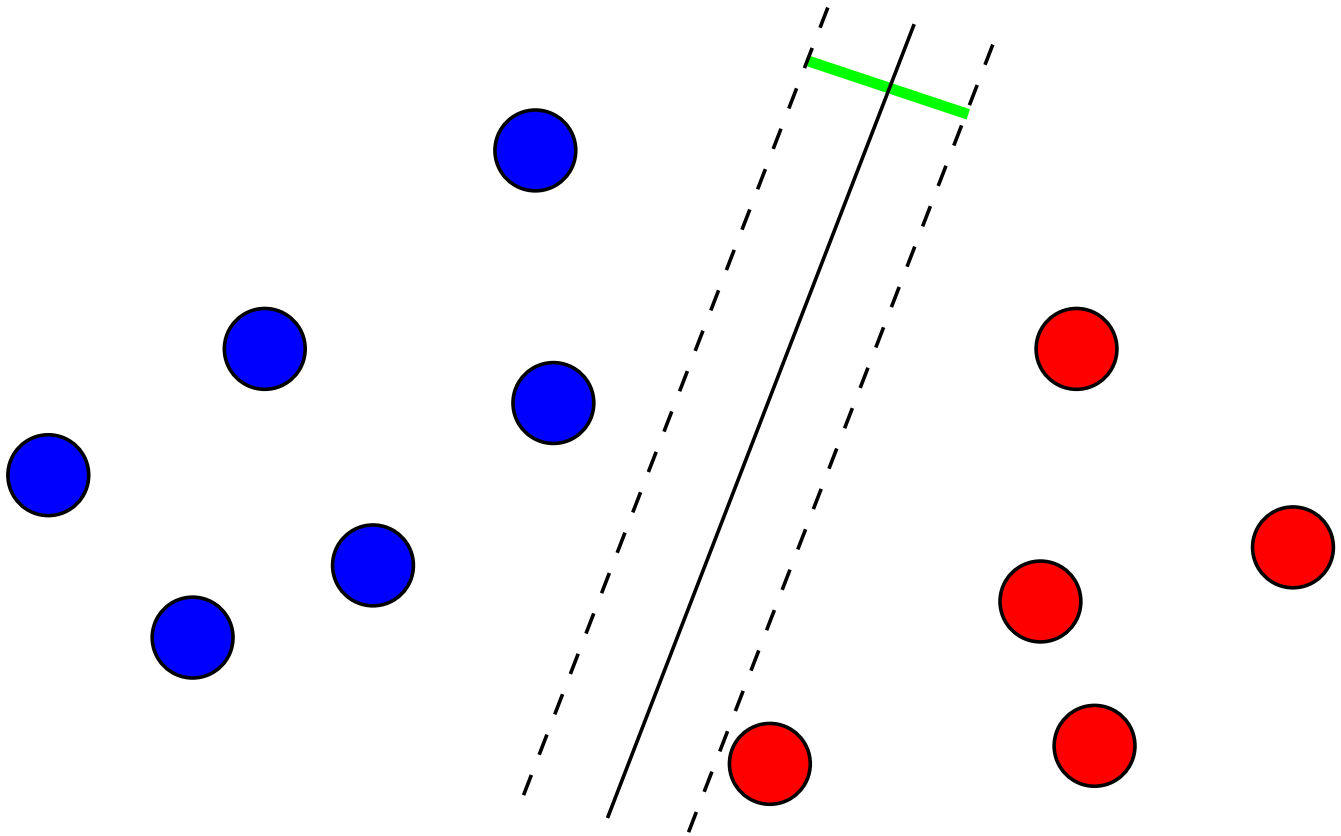
Which one is better?



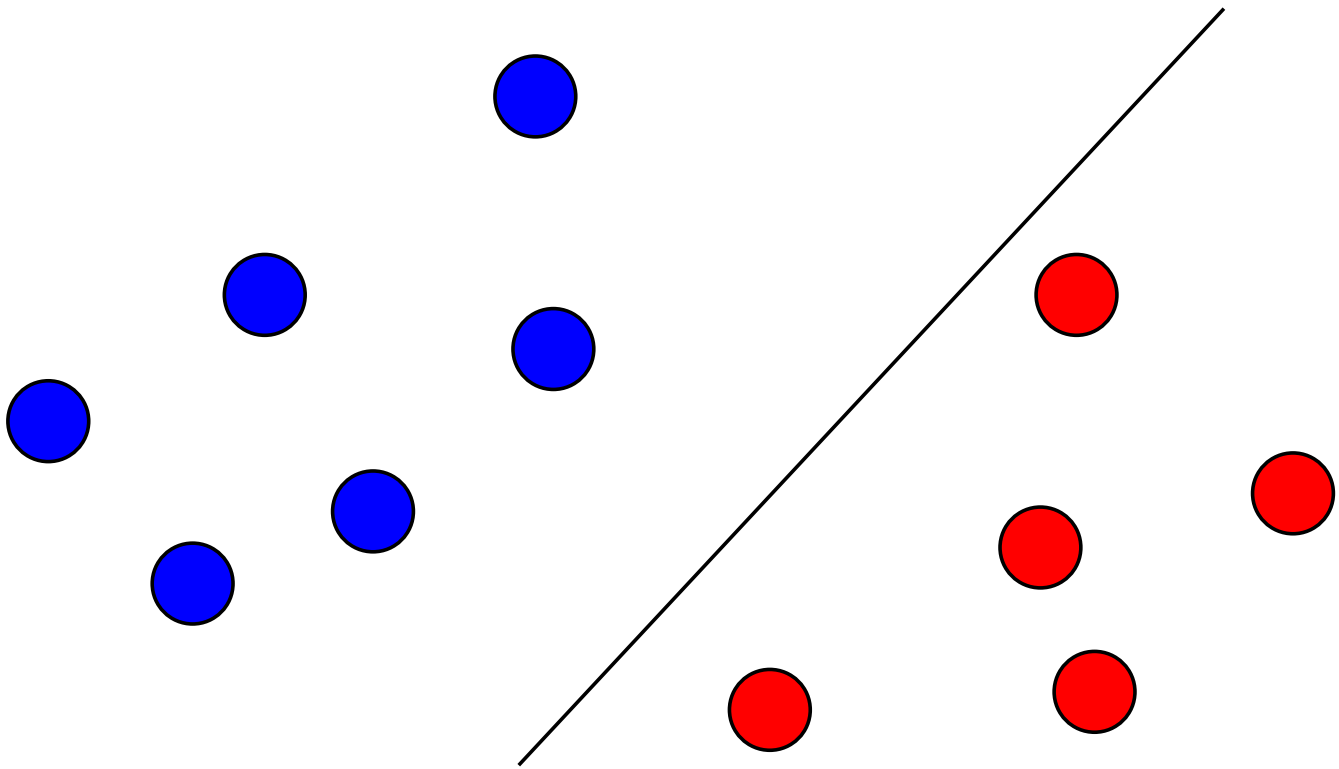
The margin of a linear classifier



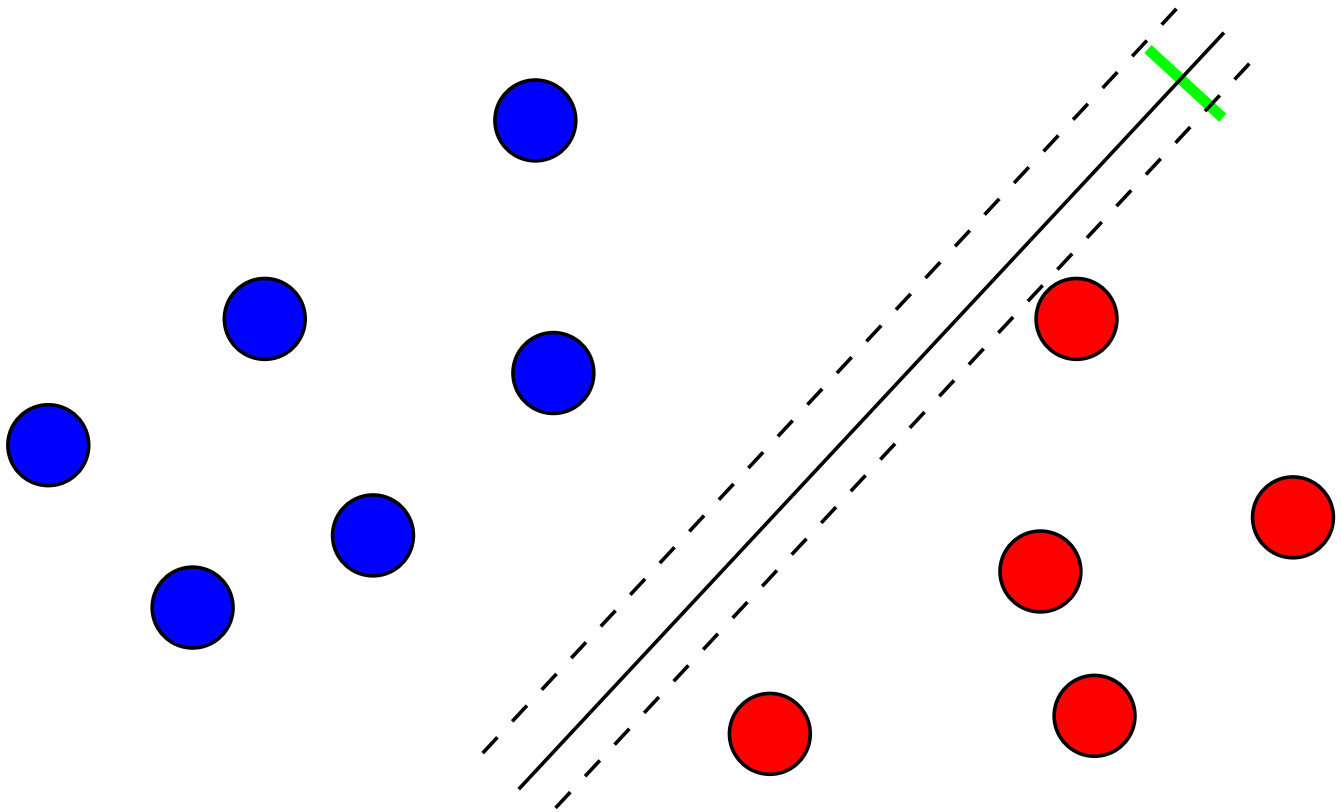
The margin of a linear classifier



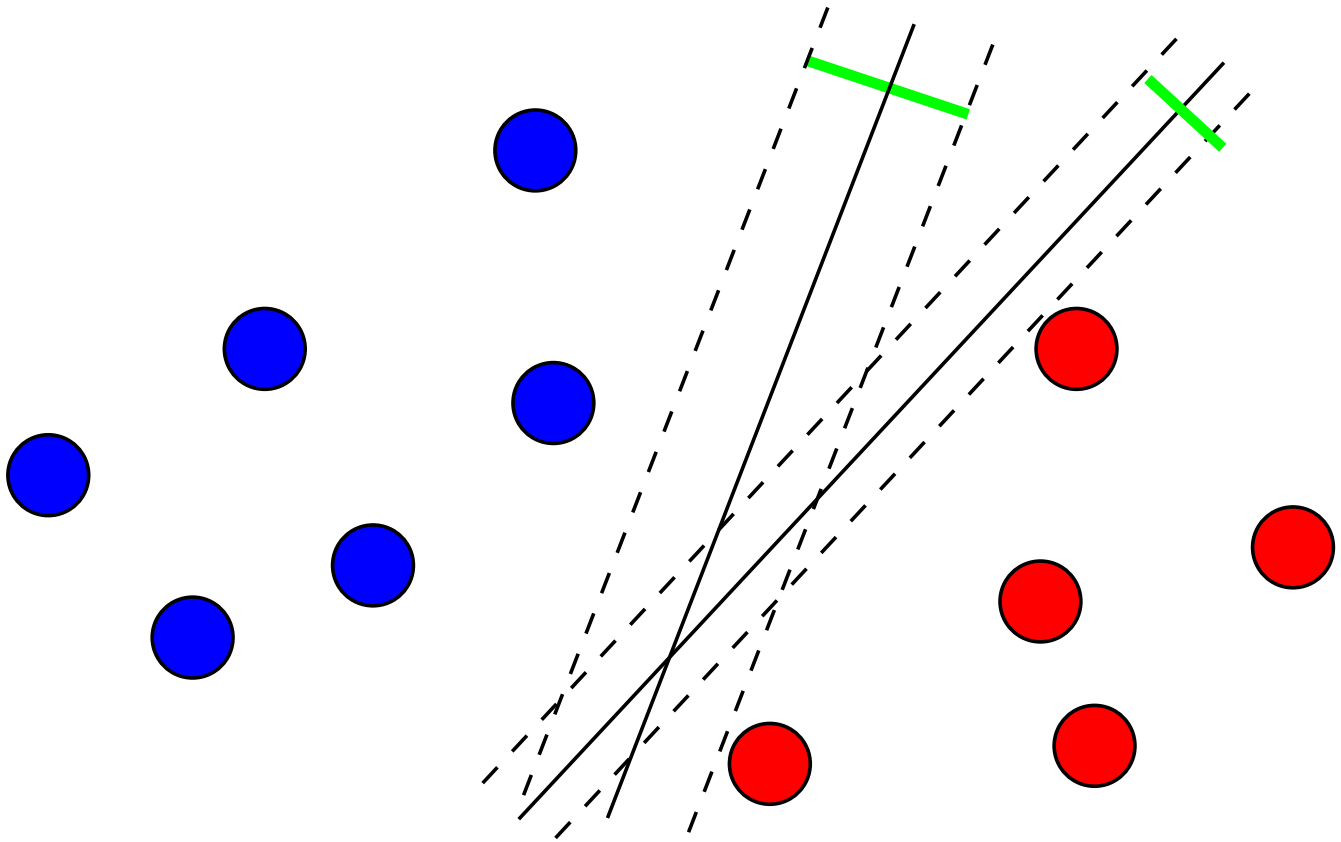
The margin of a linear classifier



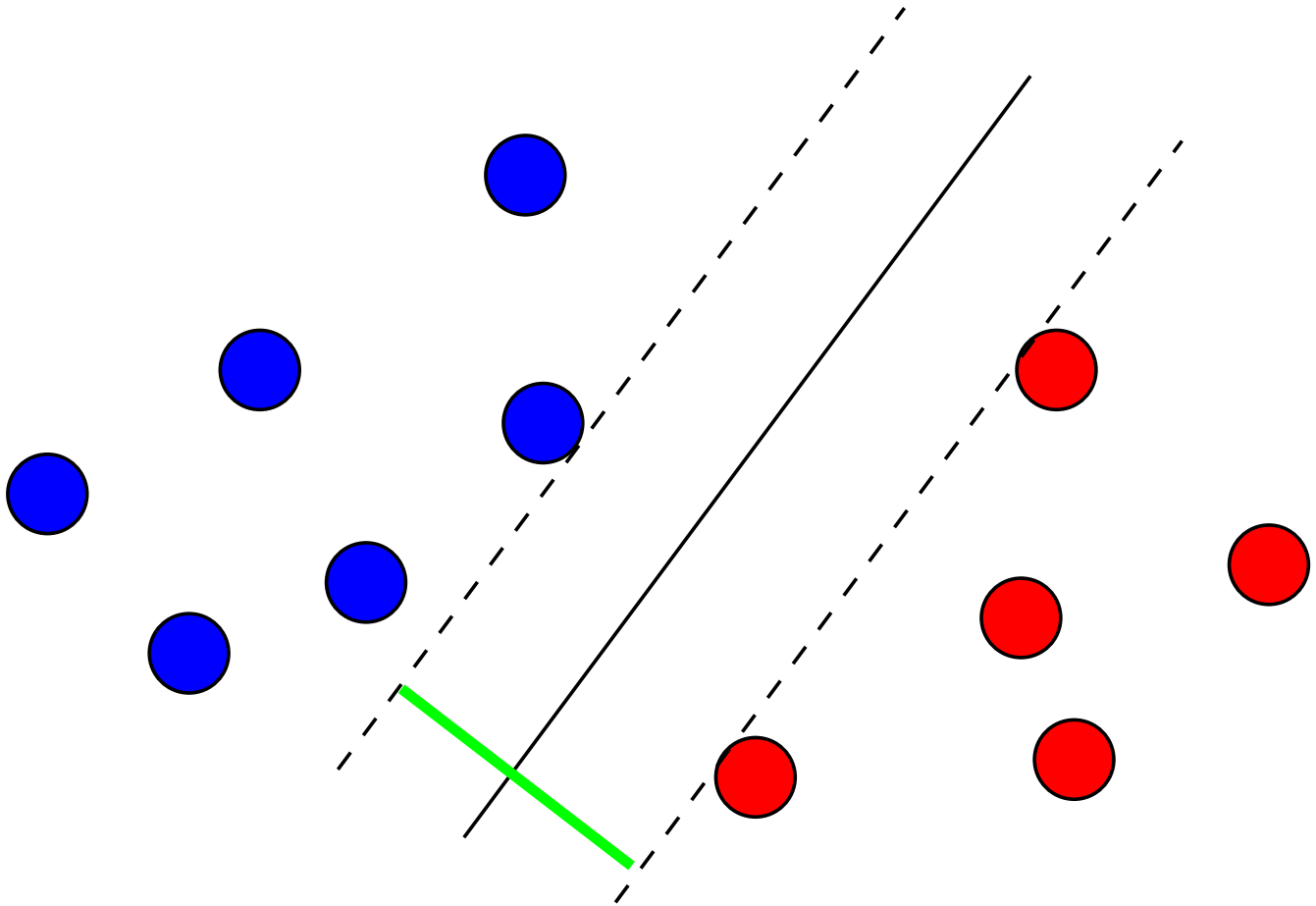
The margin of a linear classifier



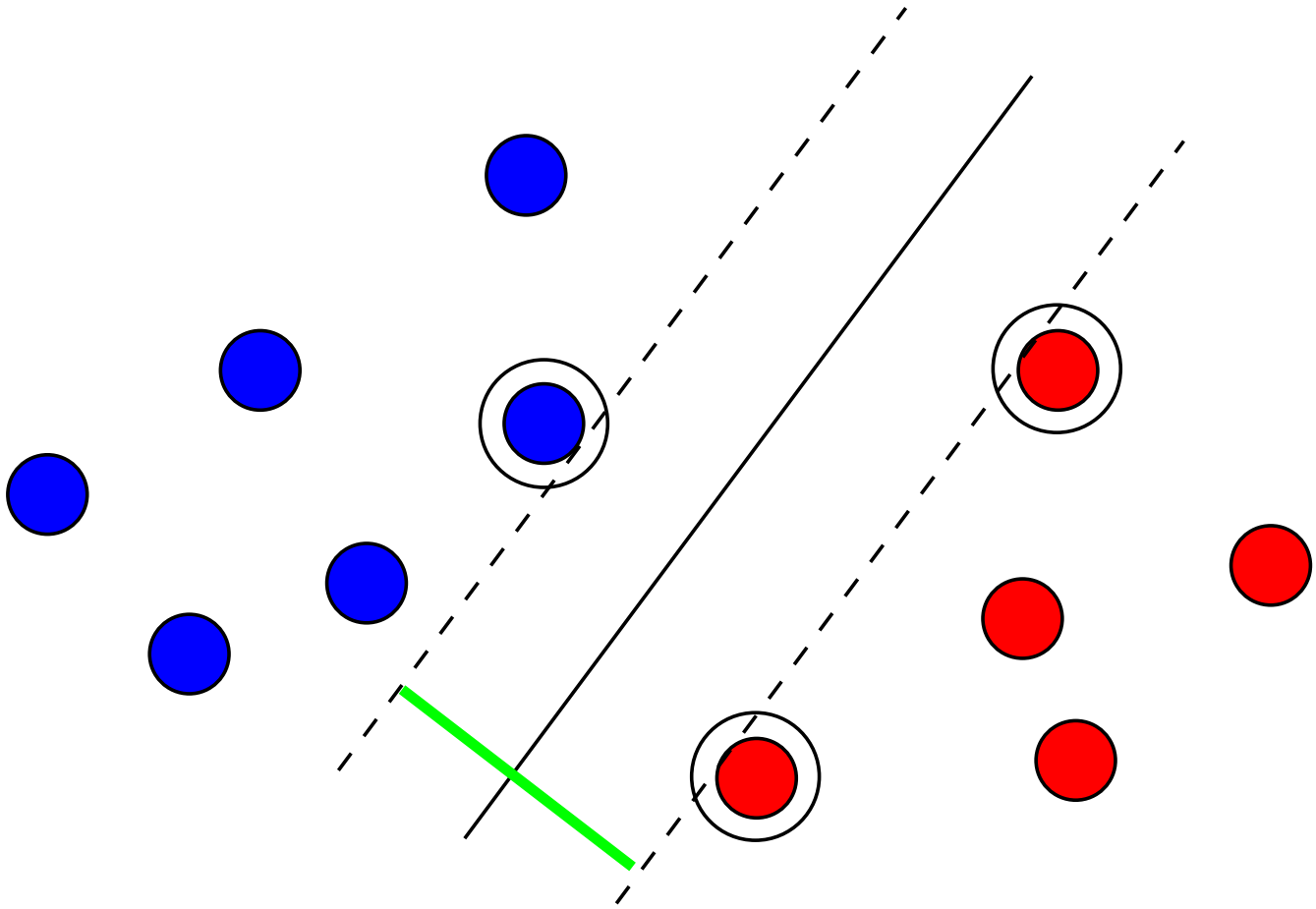
The margin of a linear classifier



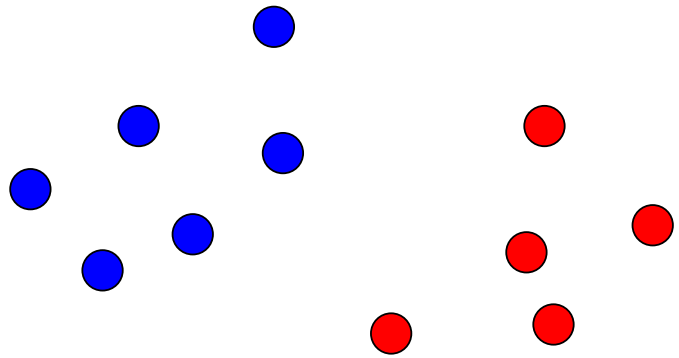
Largest margin classifier (support vector machines)



Support vectors



More formally



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

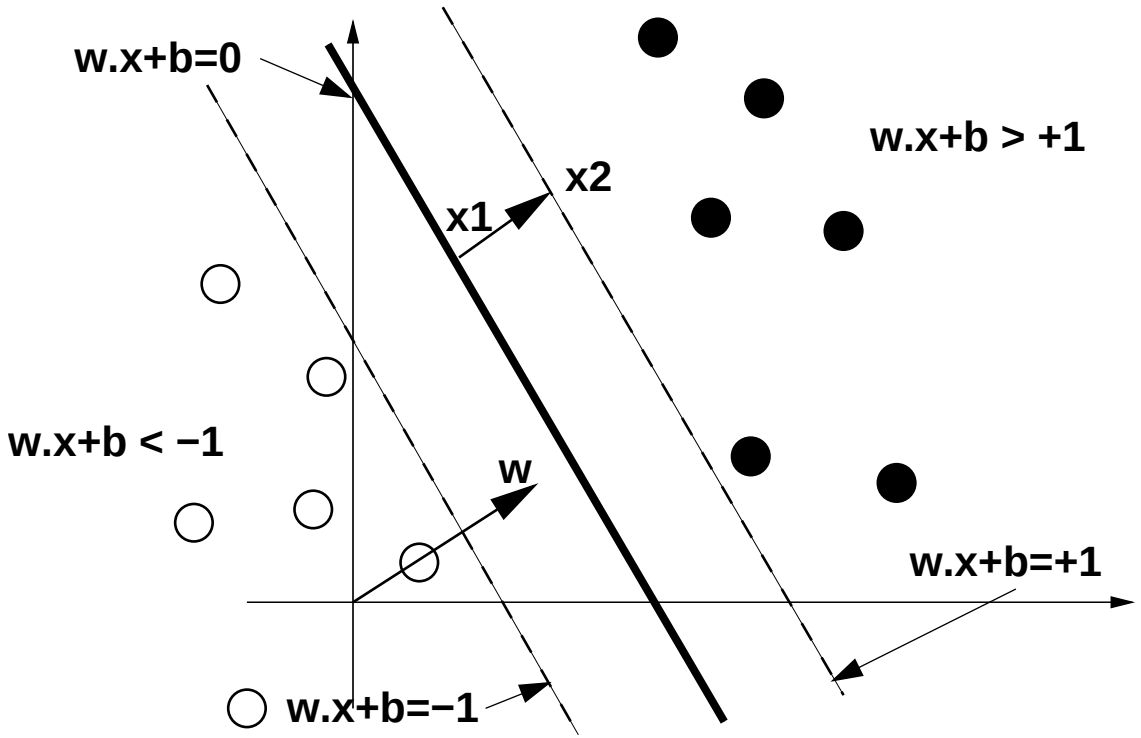
$$\mathcal{S} = \left\{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\right\} ,$$

  where $\vec{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$.
- We assume (for the moment) that the data are linearly separable, i.e., that there exists $(\vec{w}, b) \in \mathbb{R}^d \times \mathbb{R}$ 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 right side of the dotted line

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

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

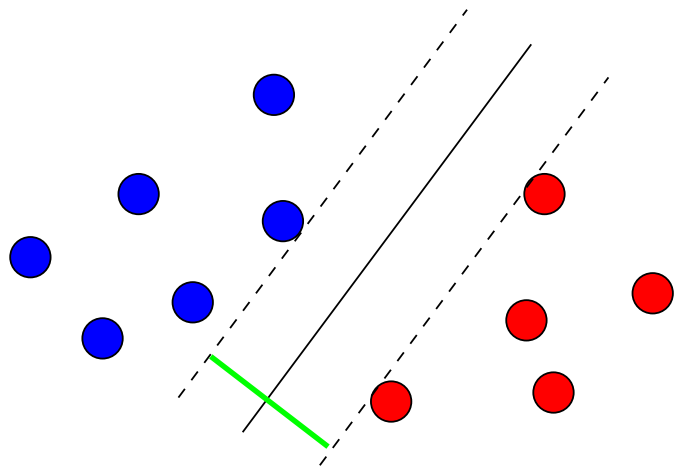
For negative examples ($y_i = -1$) this means:

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

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 $\mathbb{R}^{d+1}$.

Lagrangian

In order to minimize:

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

under the constraints:

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

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

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

Dual problem

Find $\alpha^* \in \mathbb{R}^N$ which maximizes

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

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

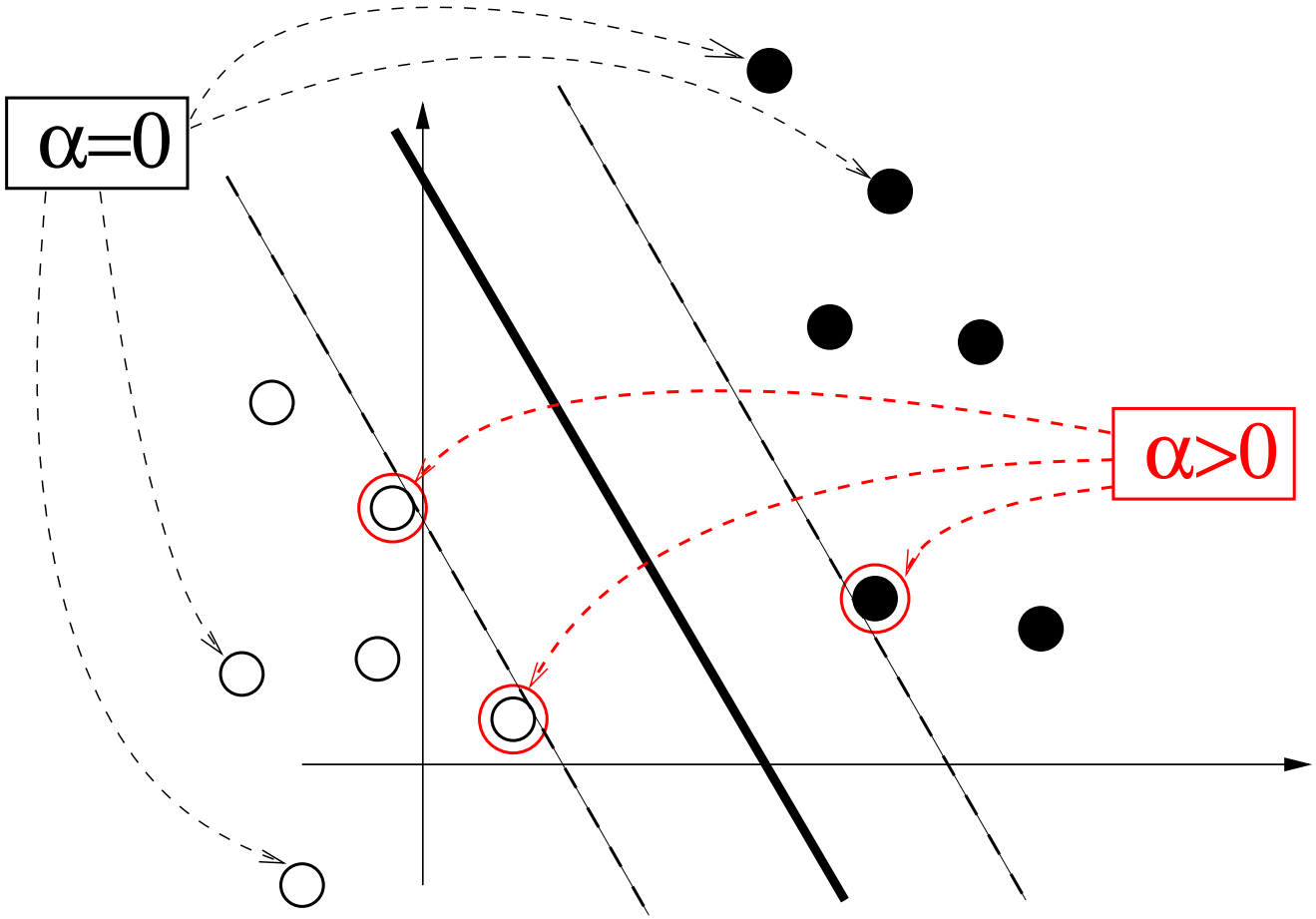
Once $\vec{\alpha}^*$ is found, we recover $(\vec{w}^*, b^*)$ corresponding to the optimal hyperplane. w^* is given by:

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

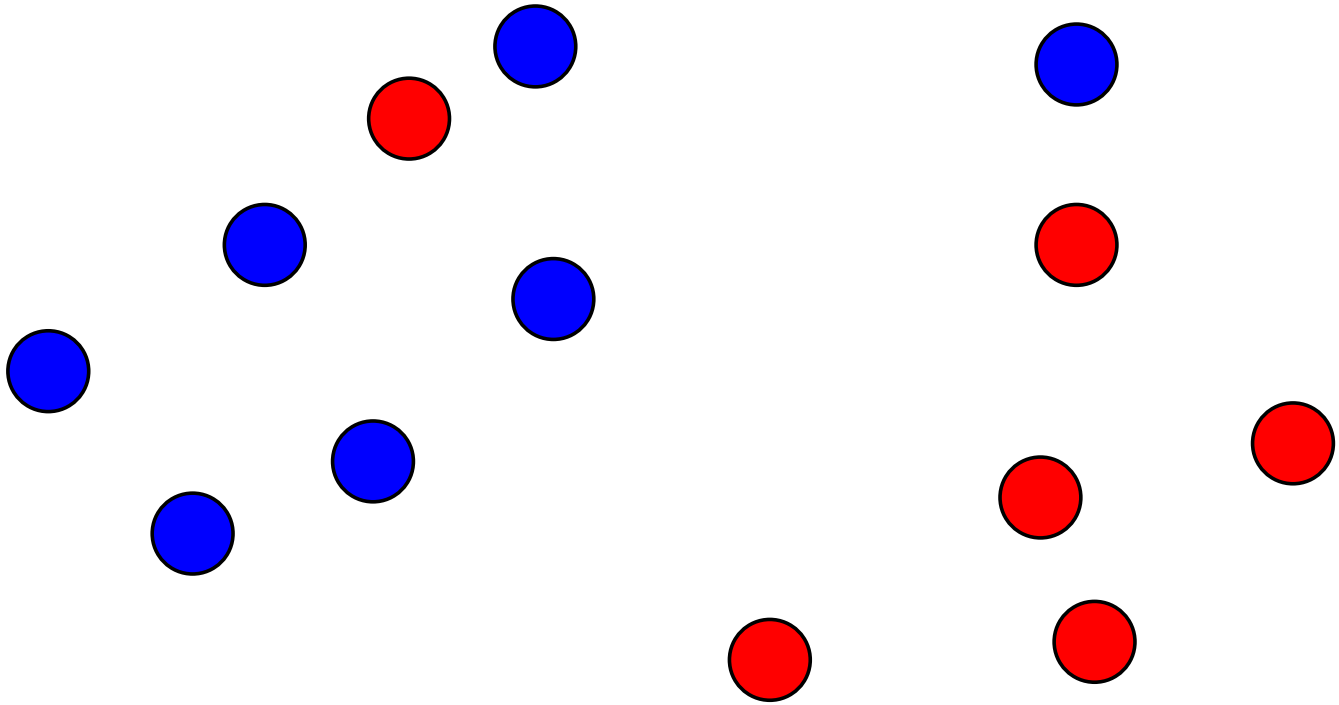
and the decision function is therefore:

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

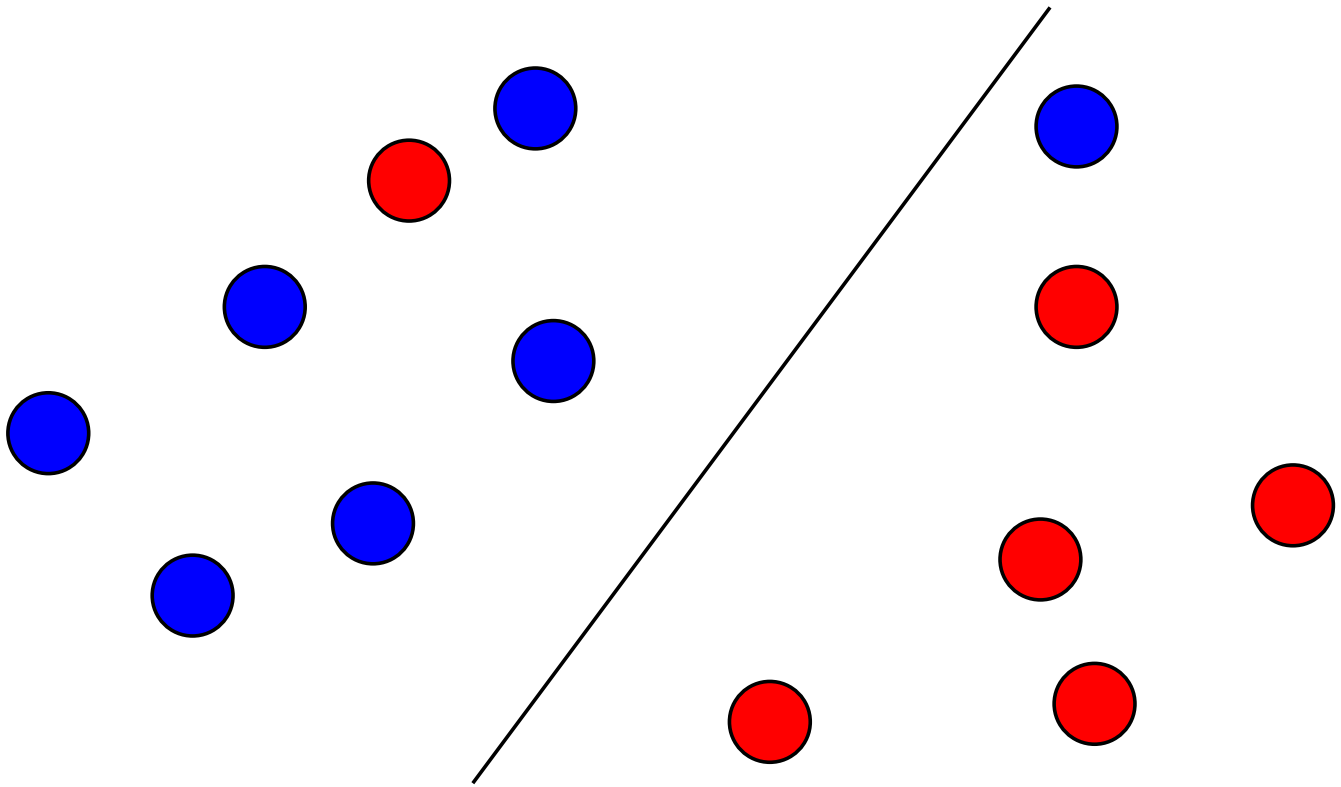
Interpretation: support vectors



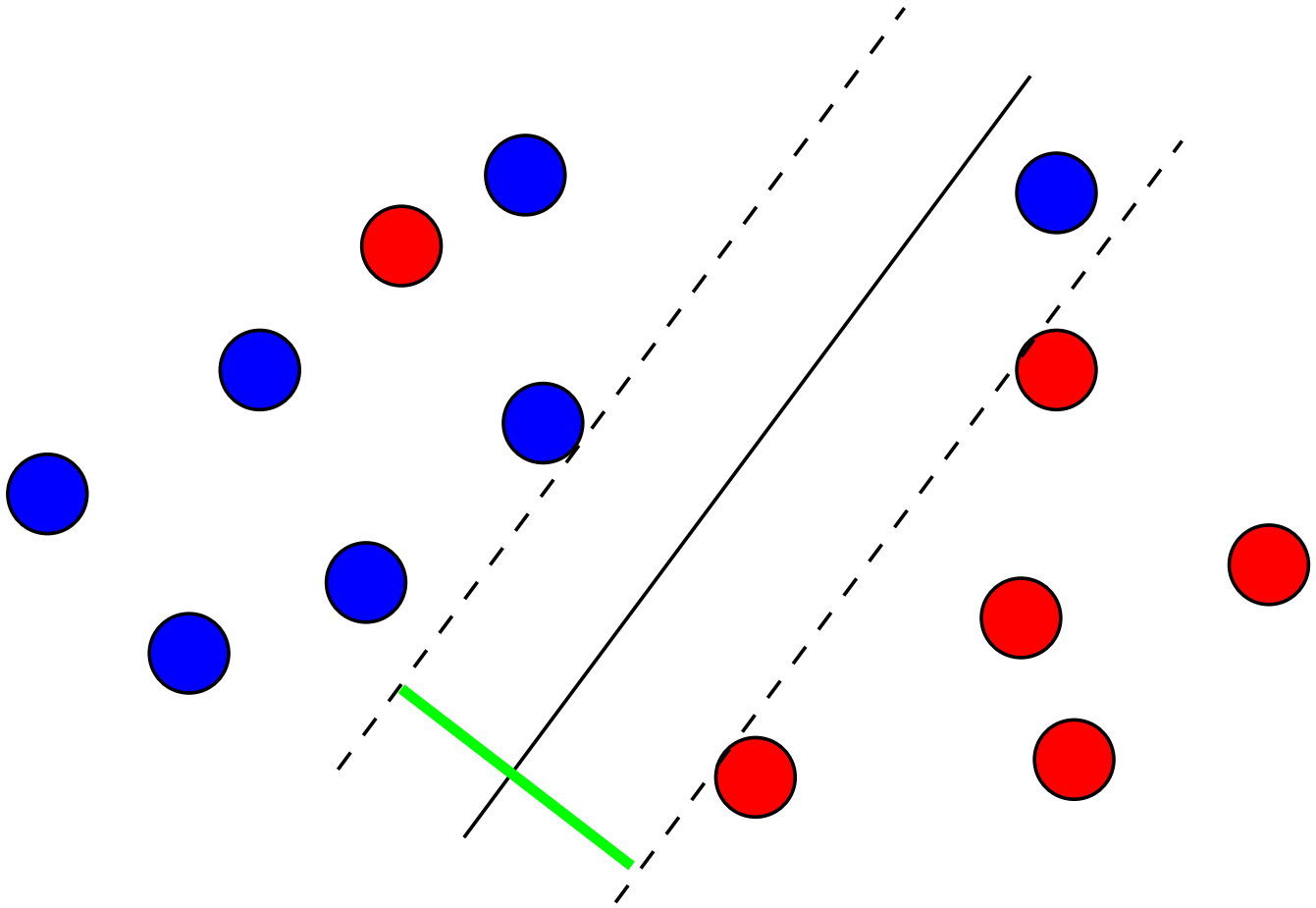
What if data are not linearly separable?



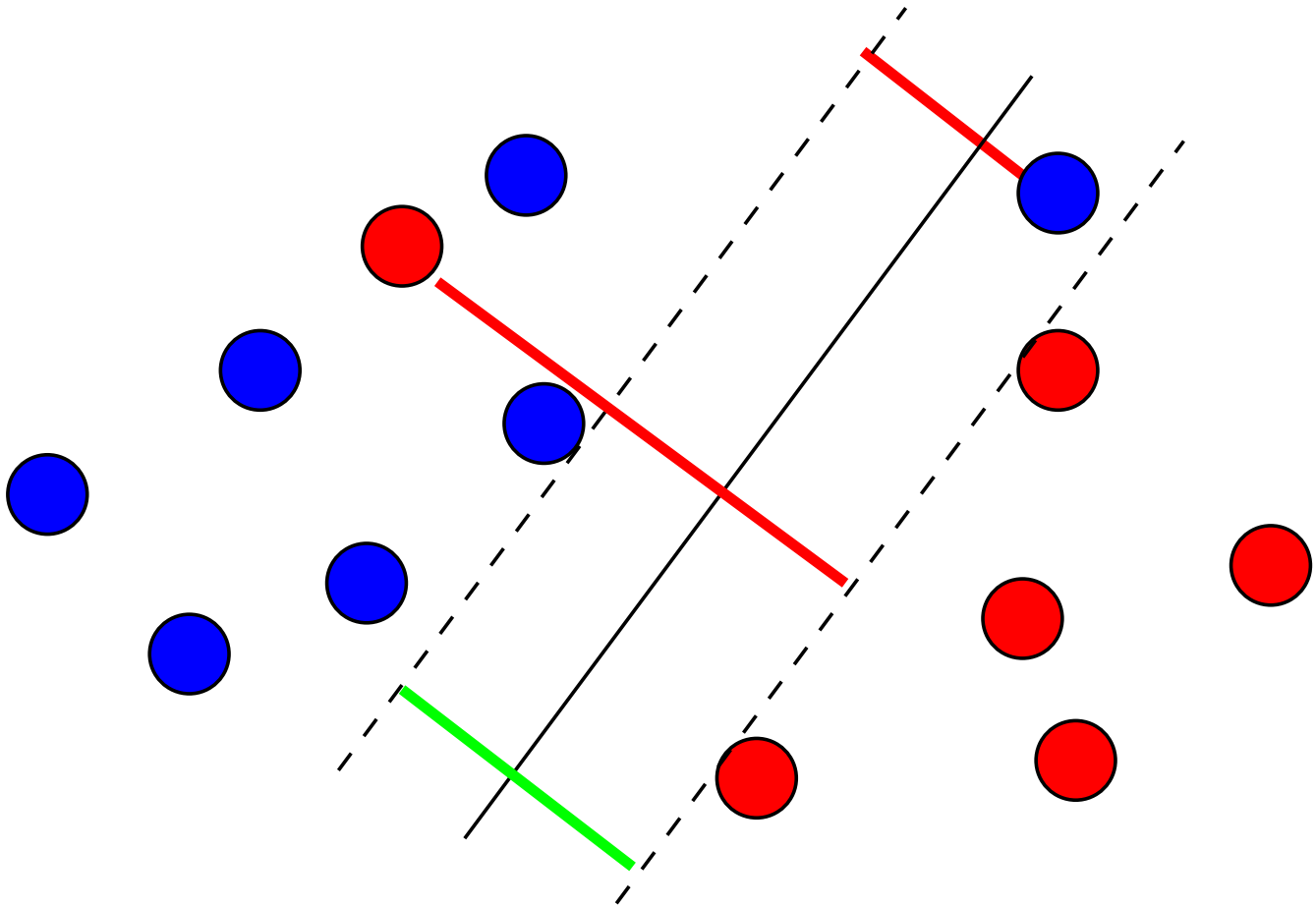
What if data are not linearly separable?



What if data are not linearly separable?



What if data are not linearly separable?

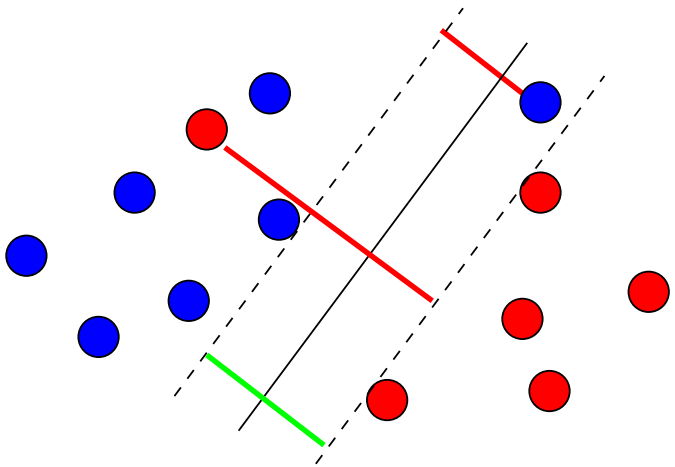


Soft-margin SVM

- 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



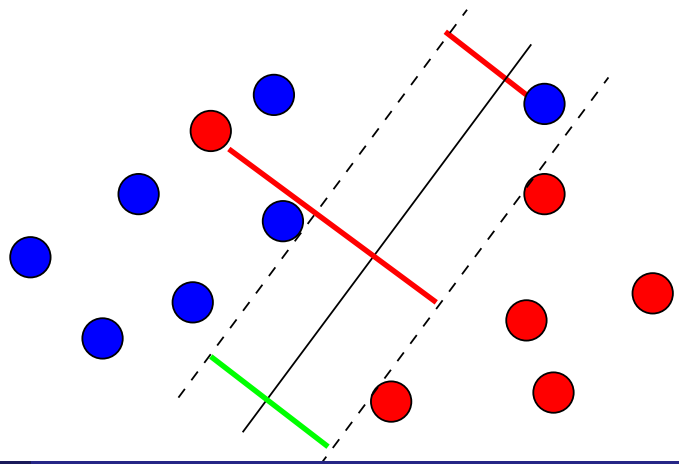
Soft-margin SVM formulation

- 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:

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



Dual formulation of soft-margin SVM

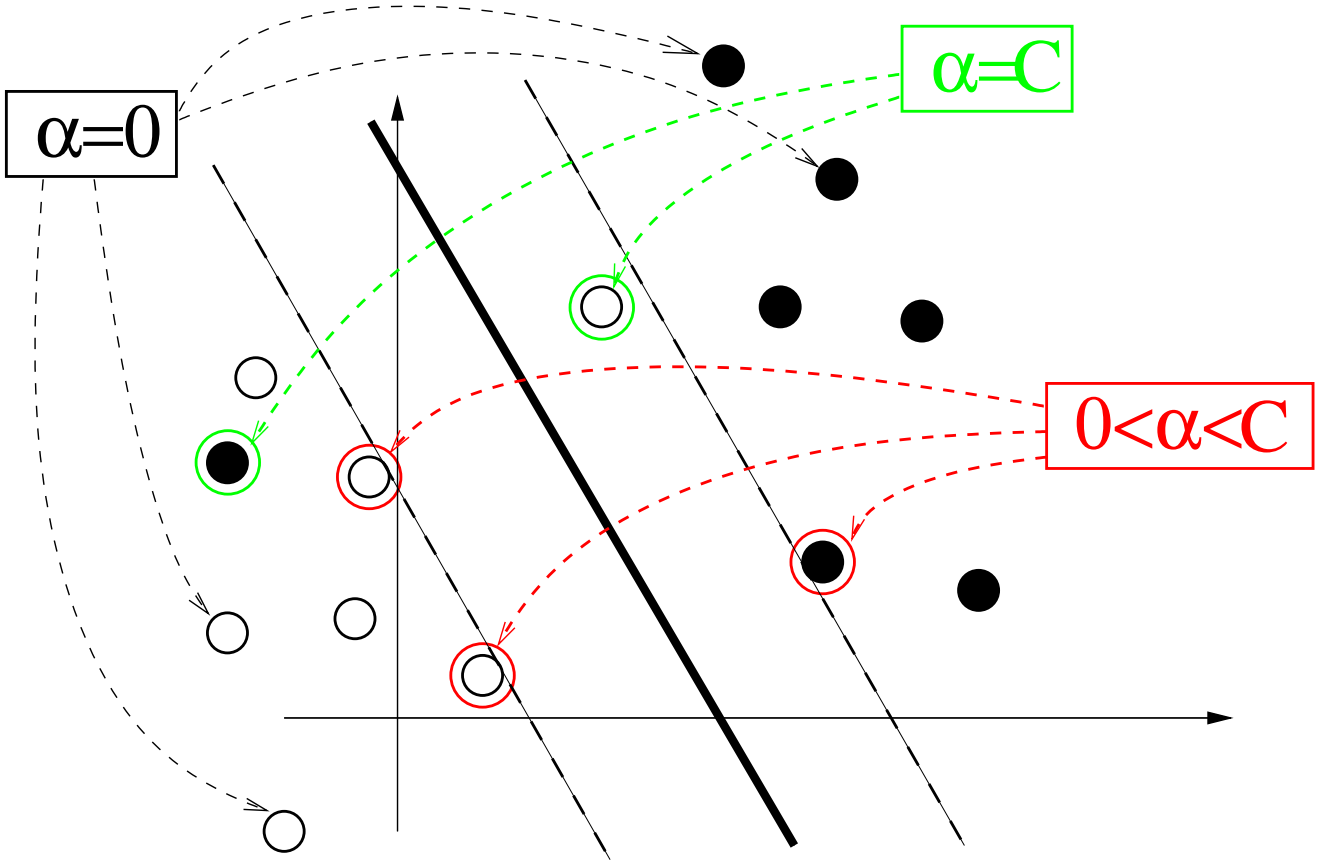
Maximize

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

under the constraints:

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

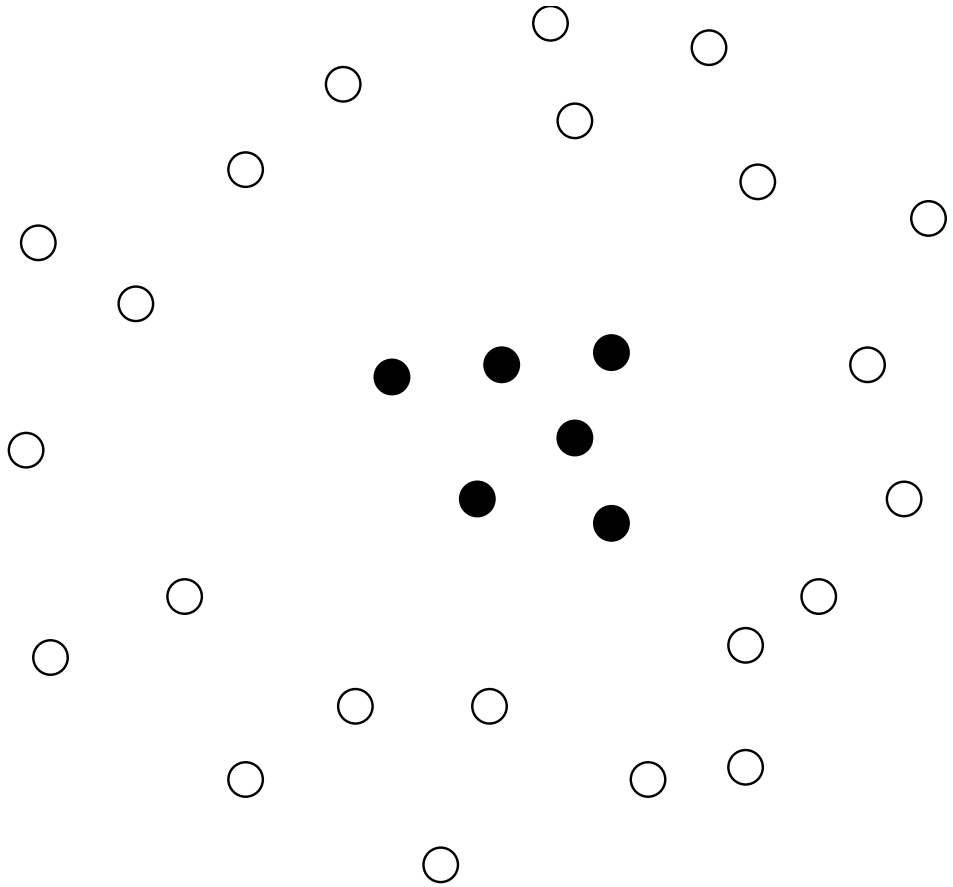
Interpretation: bounded and unbounded support vectors



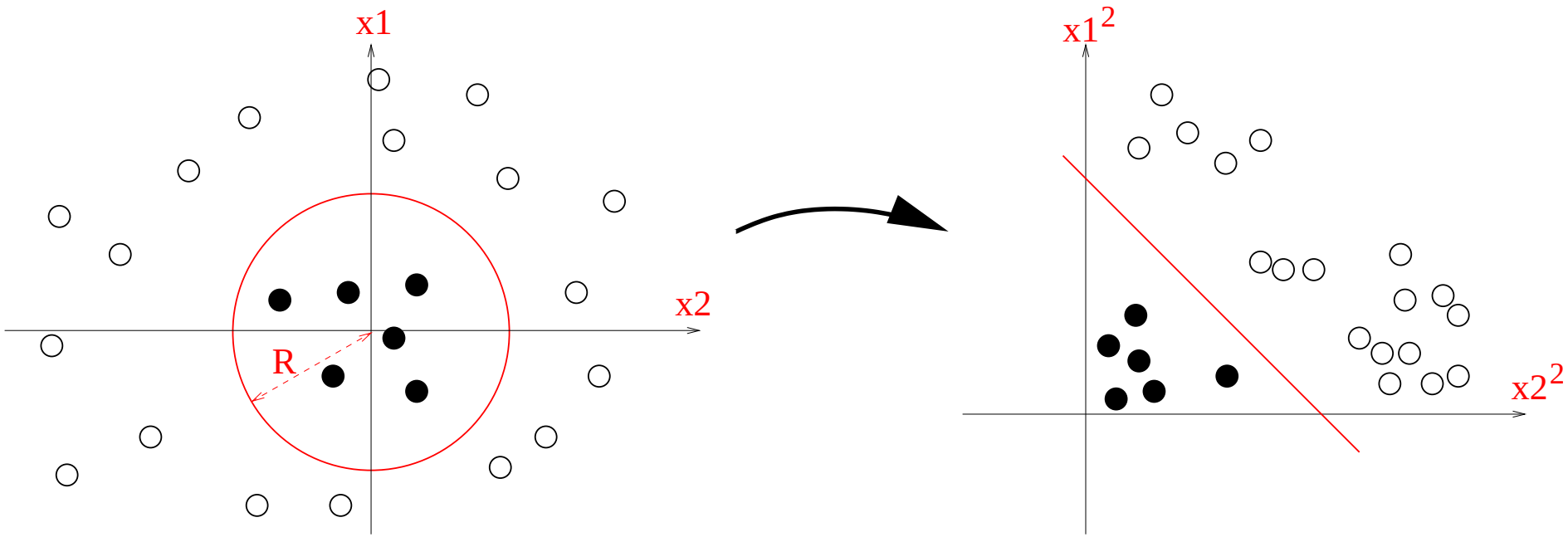
Outline

1. Machine learning in bioinformatics
2. Linear support vector machines
3. Nonlinear SVM and kernels
4. SVM for complex data: the case of graphs
5. Conclusion

Sometimes linear classifiers are not interesting



Solution: non-linear mapping to a feature space



Let $\vec{\Phi}(\vec{x}) = (x_1^2, x_2^2)'$, $\vec{w} = (1,1)'$ and $b = 1$. Then the decision function is:

$$f(\vec{x}) = x_1^2 + x_2^2 - R^2 = \vec{w}.\vec{\Phi}(\vec{x}) + b,$$

Kernel (*simple but important*)

For a given mapping Φ from the space of objects $\mathcal{X}$ to some feature space, the kernel of two objects x and x' is the inner product of their images in the features space:

$$\forall x, x' \in \mathcal{X}, \quad K(x, x') = \vec{\Phi}(x).\vec{\Phi}(x').$$

Example: if $\vec{\Phi}(\vec{x}) = (x_1^2, x_2^2)'$, *then*

$$K(\vec{x}, \vec{x'}) = \vec{\Phi}(\vec{x}).\vec{\Phi}(\vec{x'}) = (x_1)^2(x_1')^2 + (x_2)^2(x_2')^2.$$

Training a SVM in the feature space

Replace each $\vec{x}.\vec{x}'$ in the SVM algorithm by $\vec{\Phi}(x).\vec{\Phi}(x') = K(x, x')$
The dual problem is to maximize

$$L(\vec{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i, x_j),$$

under the constraints:

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

Predicting with a SVM in the feature space

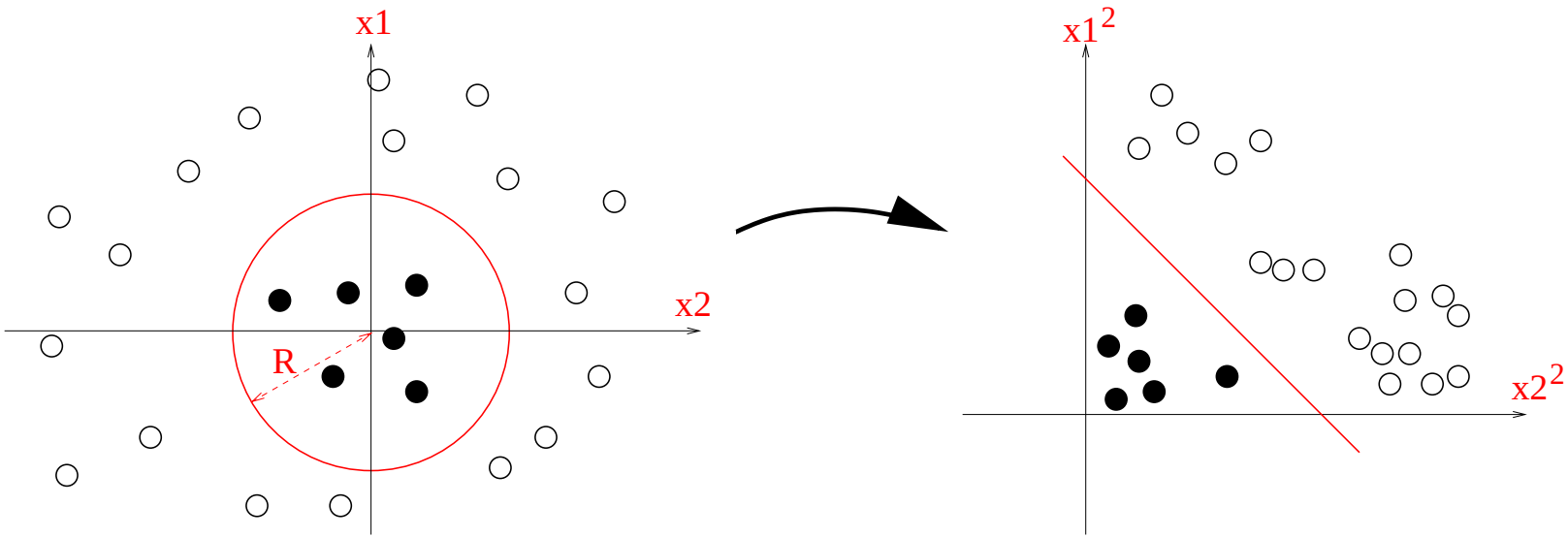
The decision function becomes:

$$\begin{aligned} f(x) &= \vec{w}^* . \vec{\Phi}(x) + b^* \\ &= \sum_{i=1}^{N} \alpha_i K(x_i, x) + b^* . \end{aligned} \tag{2}$$

The kernel trick

- The explicit computation of $\vec{\Phi}(x)$ is not necessary. The kernel $K(x, x')$ is enough. SVM work implicitly in the feature space.
- It is sometimes possible to easily compute kernels which correspond to complex large-dimensional feature spaces.

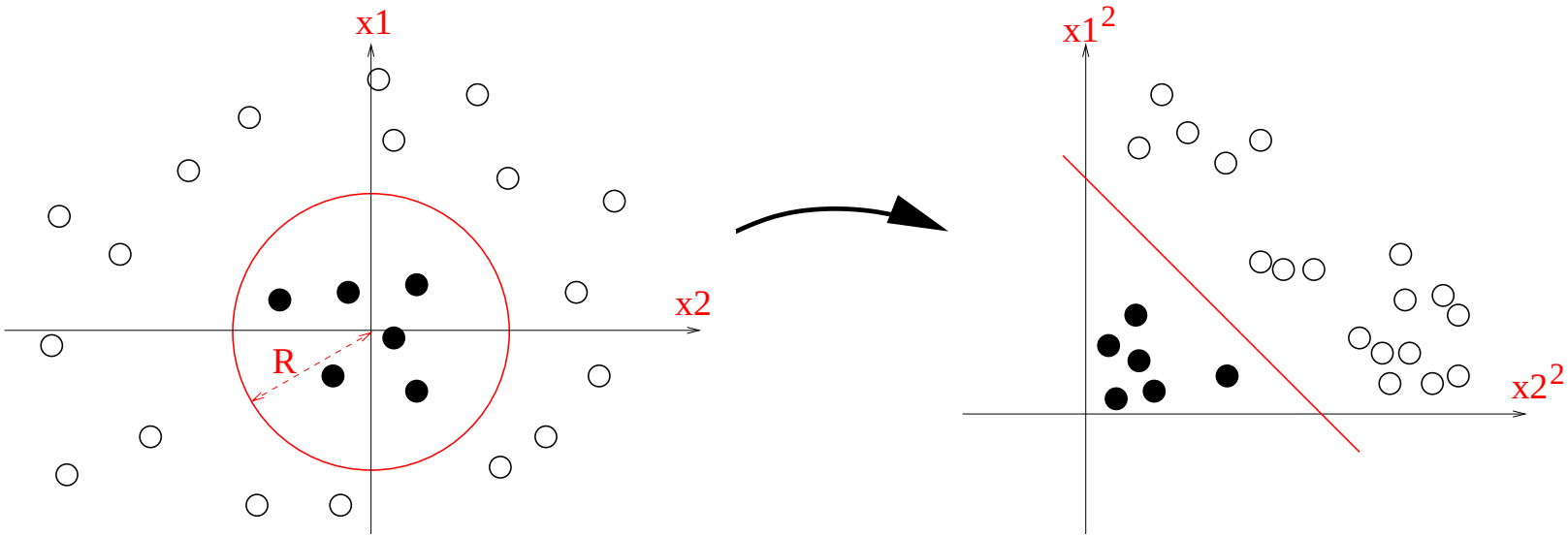
Kernel example: polynomial kernel



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

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

Kernel example: polynomial kernel



More generally,

$$K(\vec{x}, \vec{x}') = \left(\vec{x}.\vec{x}' + 1\right)^d$$

is an inner product in a feature space of all monomials of degree up to d *(left as exercice.)*

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

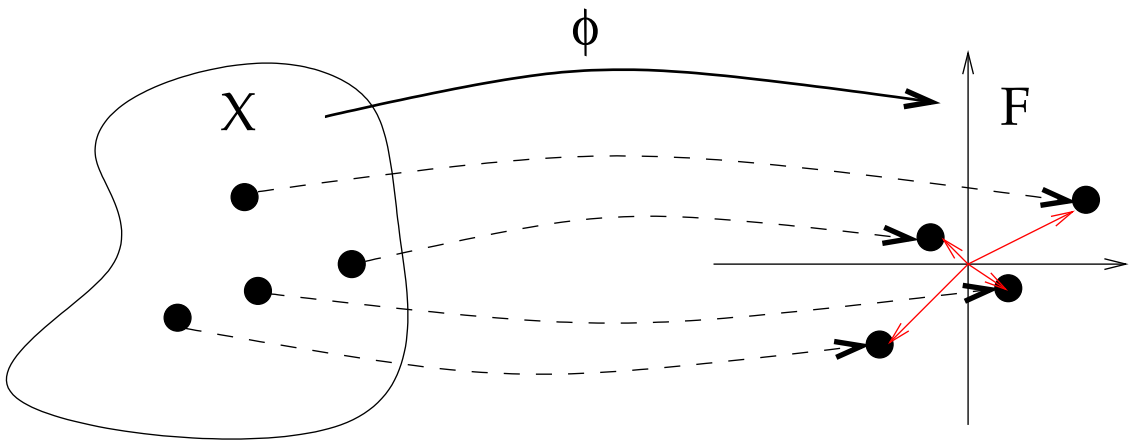
Definition

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 $\mathbf{x}, \mathbf{x}'$ in $\mathcal{X}$:

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



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:

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

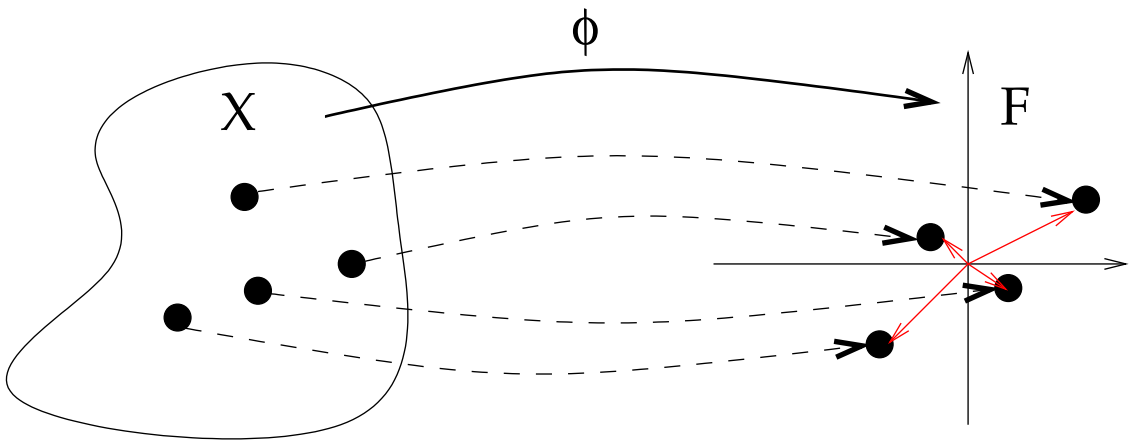
and which satisfies, for all $N \in \mathbb{N}$, $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) \in \mathcal{X}^N$ et $(a_1, a_2, \ldots, a_N) \in \mathbb{R}^N$:

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

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

Kernel examples

- Polynomial (on $\mathbb{R}^d$):

$$K(x, x') = (x.x' + 1)^d$$

- Gaussian radial basis function (RBF) (on $\mathbb{R}^d$)

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

- Laplace kernel (on $\mathbb{R}$)

$$K(x, x') = \exp\left(-\gamma|x - x'|\right)$$

- Min kernel (on $\mathbb{R}_+$)

$$K(x, x') = \min(x, x')$$

Exercice: for each kernel, find a Hilbert space $\mathcal{H}$ and a mapping $\Phi : \mathcal{X} \to \mathcal{H}$ such that $K(x, x') = \langle \Phi(x), \Phi(x') \rangle$

Example: SVM with a Gaussian kernel

- Training:

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

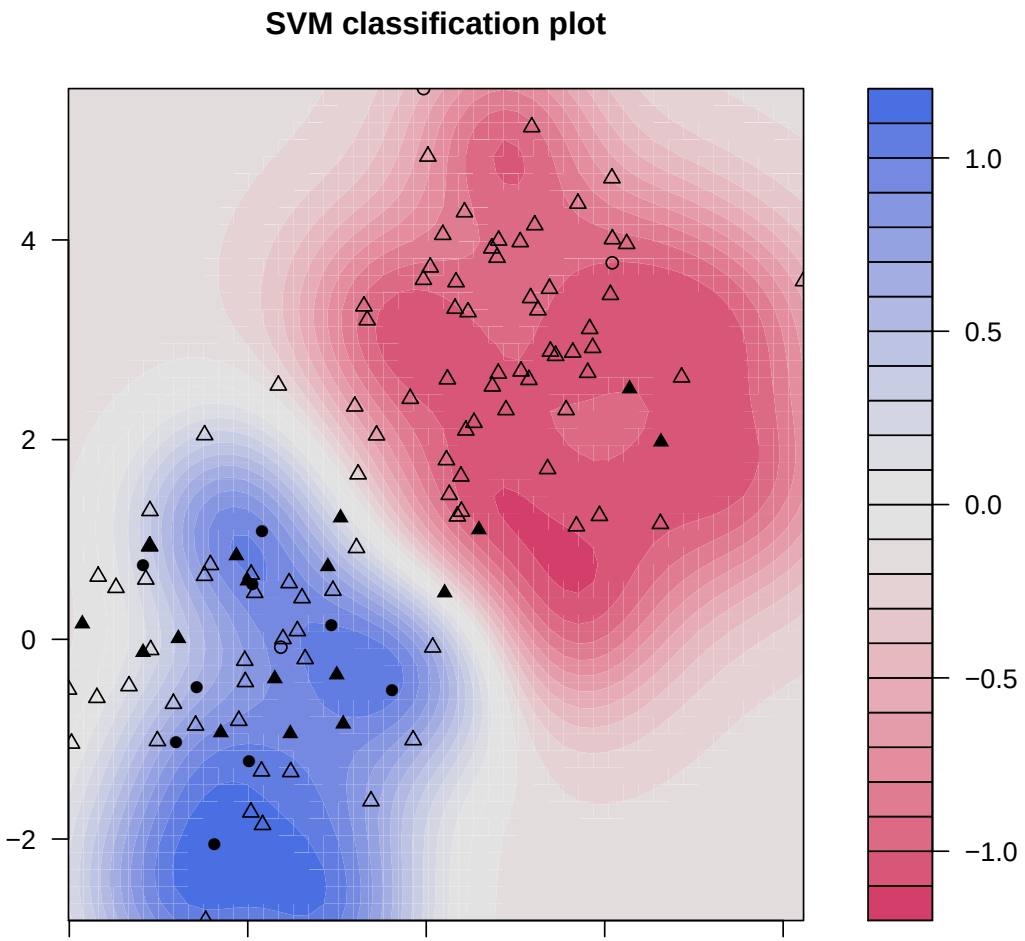
$$\text{s.t. } 0 \leq \alpha_i \leq C, \quad \text{and } \sum_{i=1}^N \alpha_i y_i = 0.$$

- Prediction

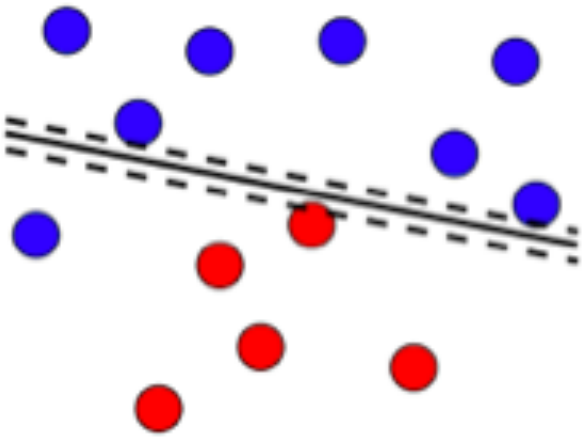
$$f(\vec{x}) = \sum_{i=1}^N \alpha_i \exp\left(-\frac{||\vec{x} - \vec{x}_i||^2}{2\sigma^2}\right)$$

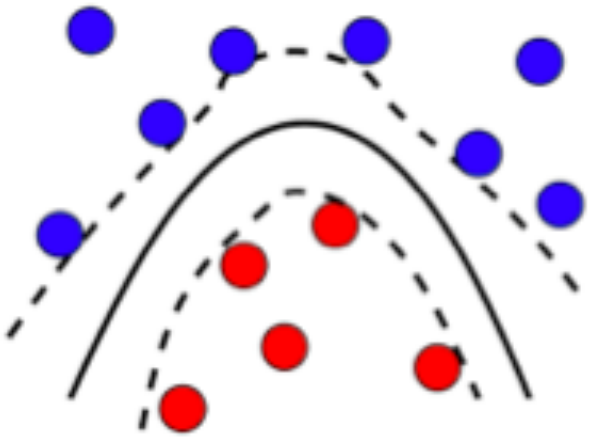
Example: SVM with a Gaussian kernel

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

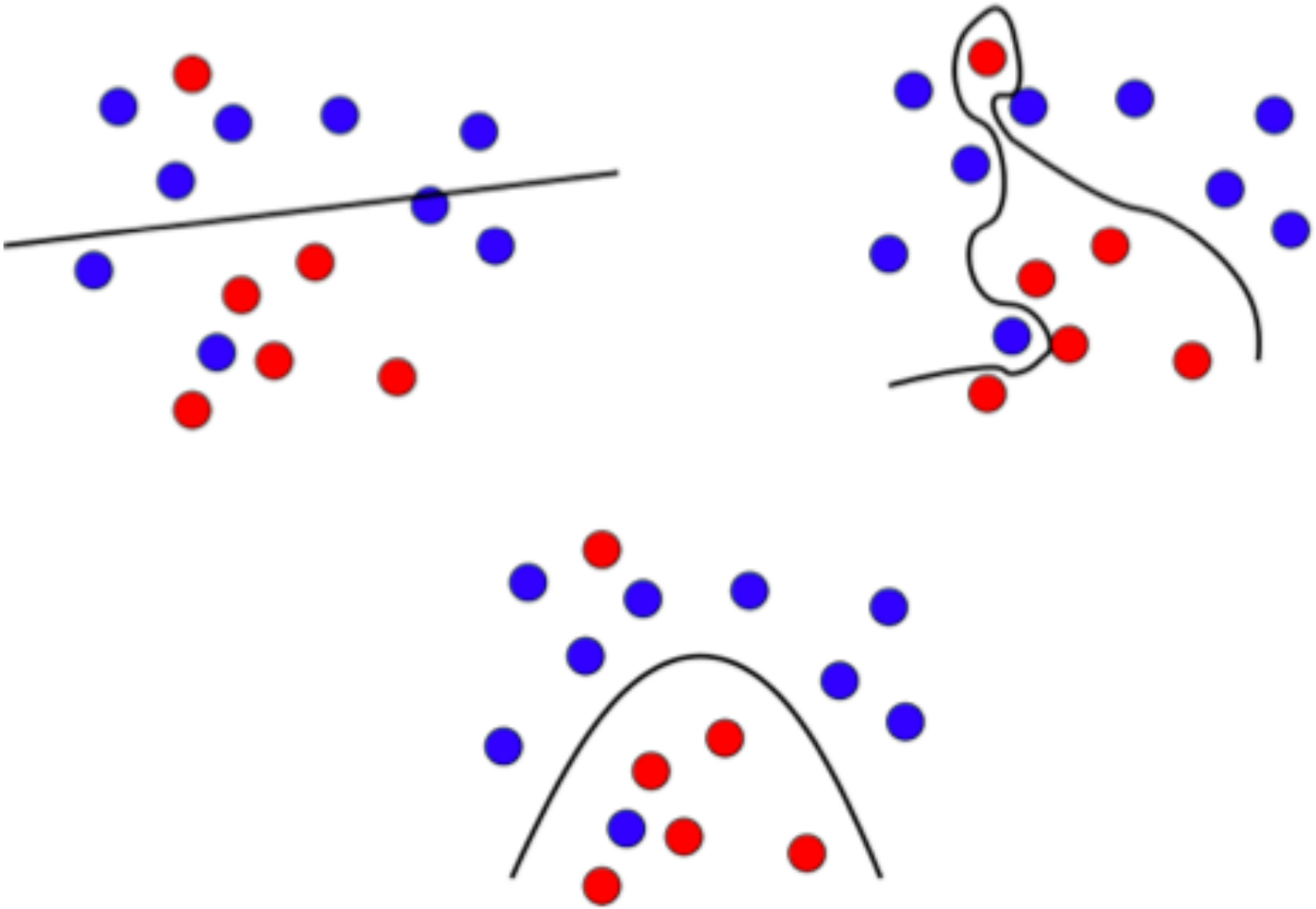


Linear vs nonlinear SVM





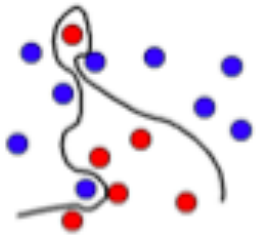
Regularity vs data fitting trade-off



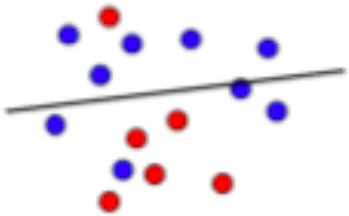
C controls the trade-off

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

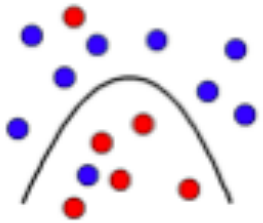
- Large C :
  - makes few errors



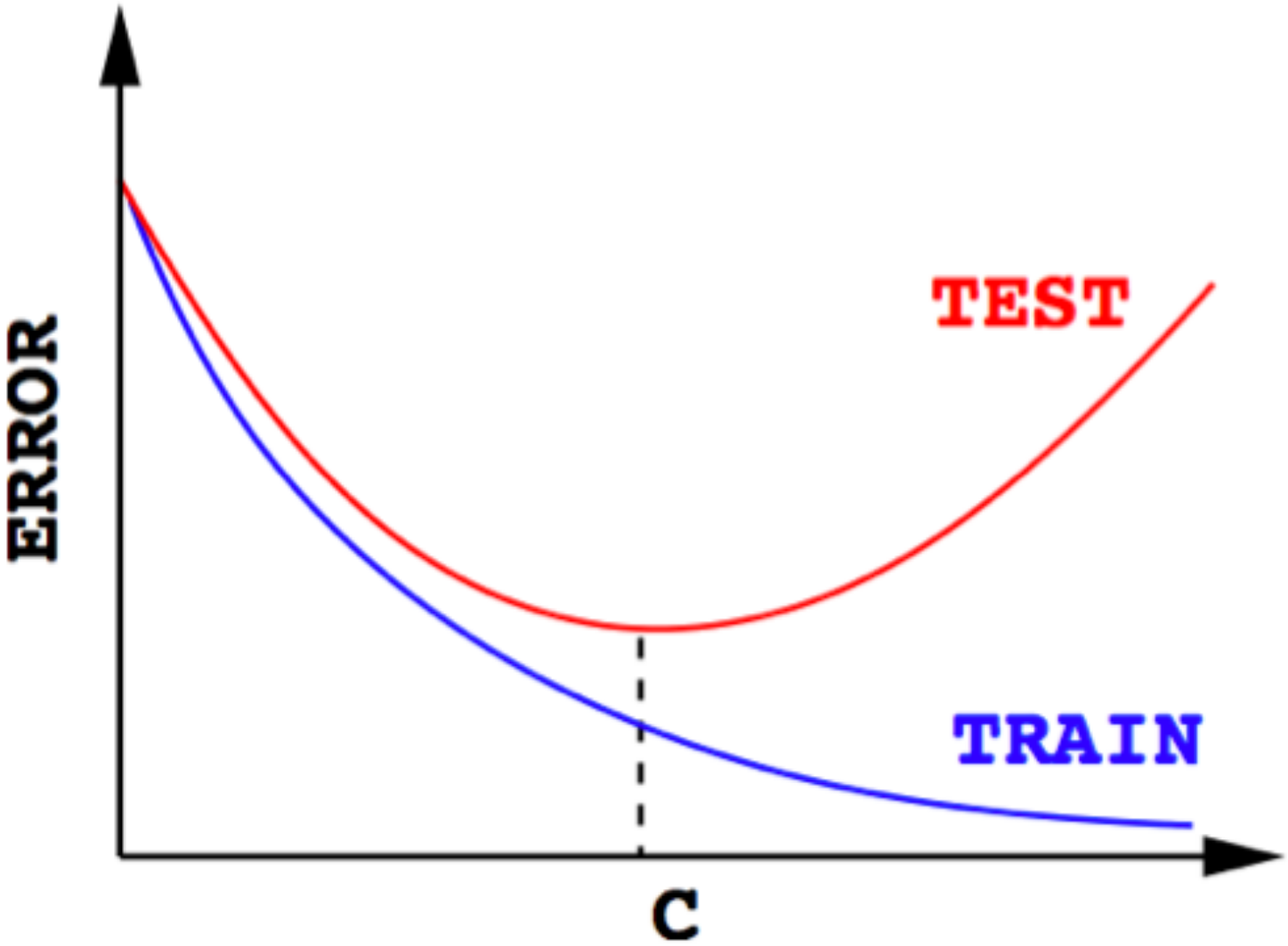
- Small C :
  - ensure a large margin



- Intermediate C:
  - finds a trade-off



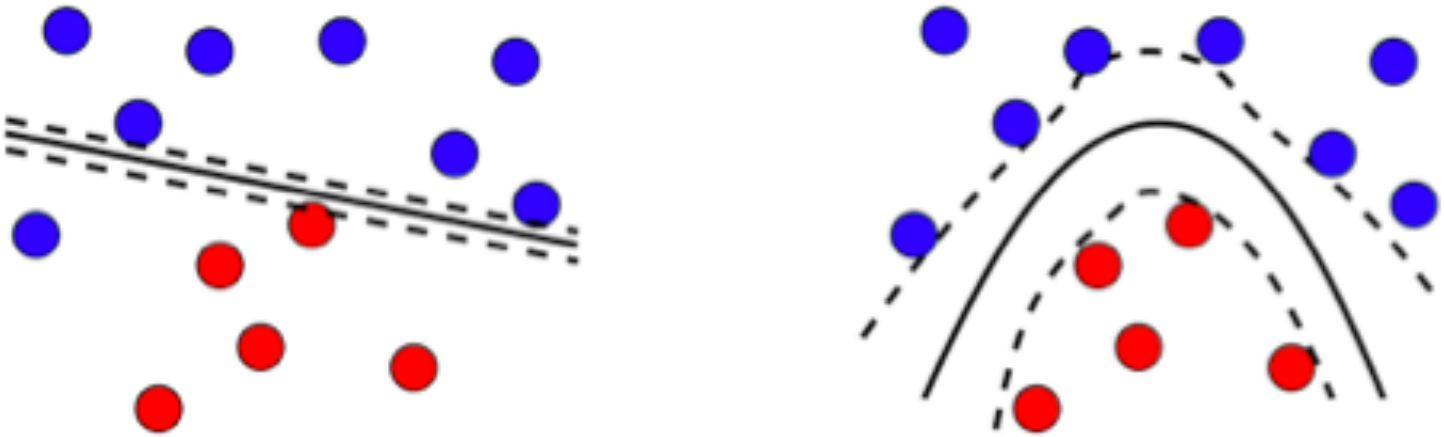
Why it is important to control the trade-off



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)

SVM summary

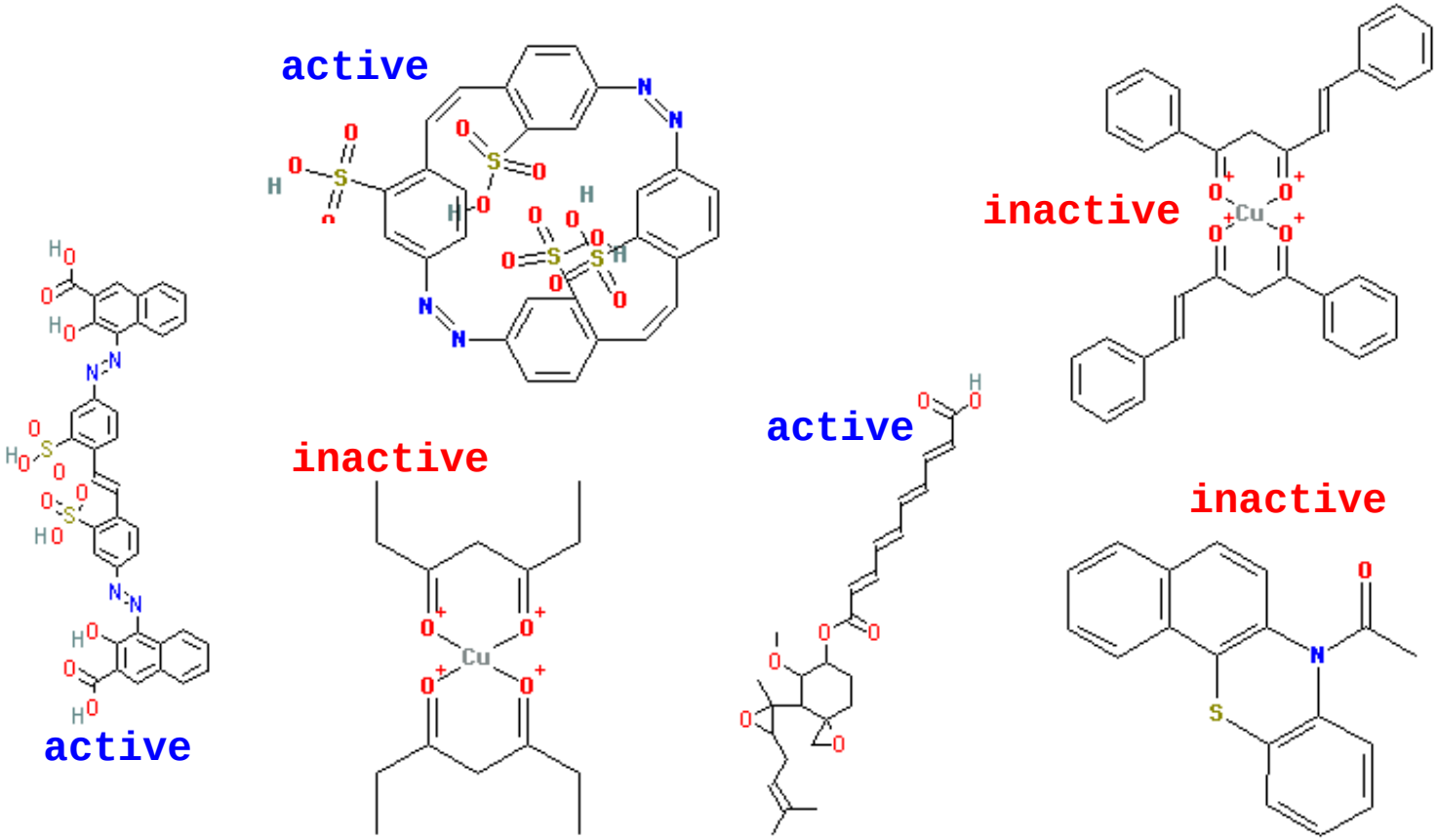


- Large margin
- Linear or nonlinear (with the kernel trick)
- Control of the regularization / data fitting trade-off with C

Outline

1. Machine learning in bioinformatics
2. Linear support vector machines
3. Nonlinear SVM and kernels
4. SVM for complex data: the case of graphs
5. Conclusion

Virtual screening for drug discovery



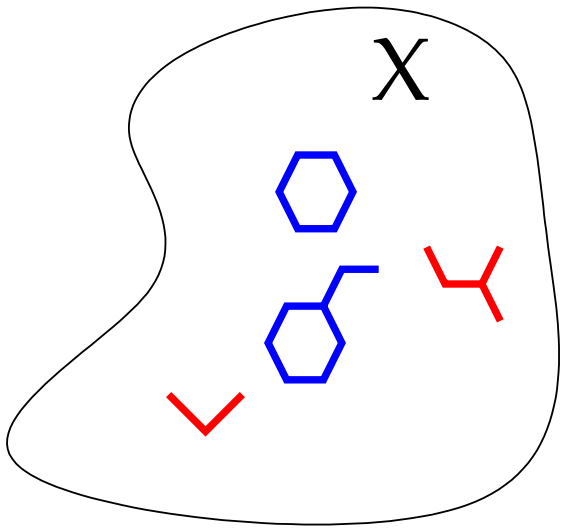
NCI AIDS screen results (from http://cactus.nci.nih.gov).

Classification with SVM

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 $\mathcal{H}$.

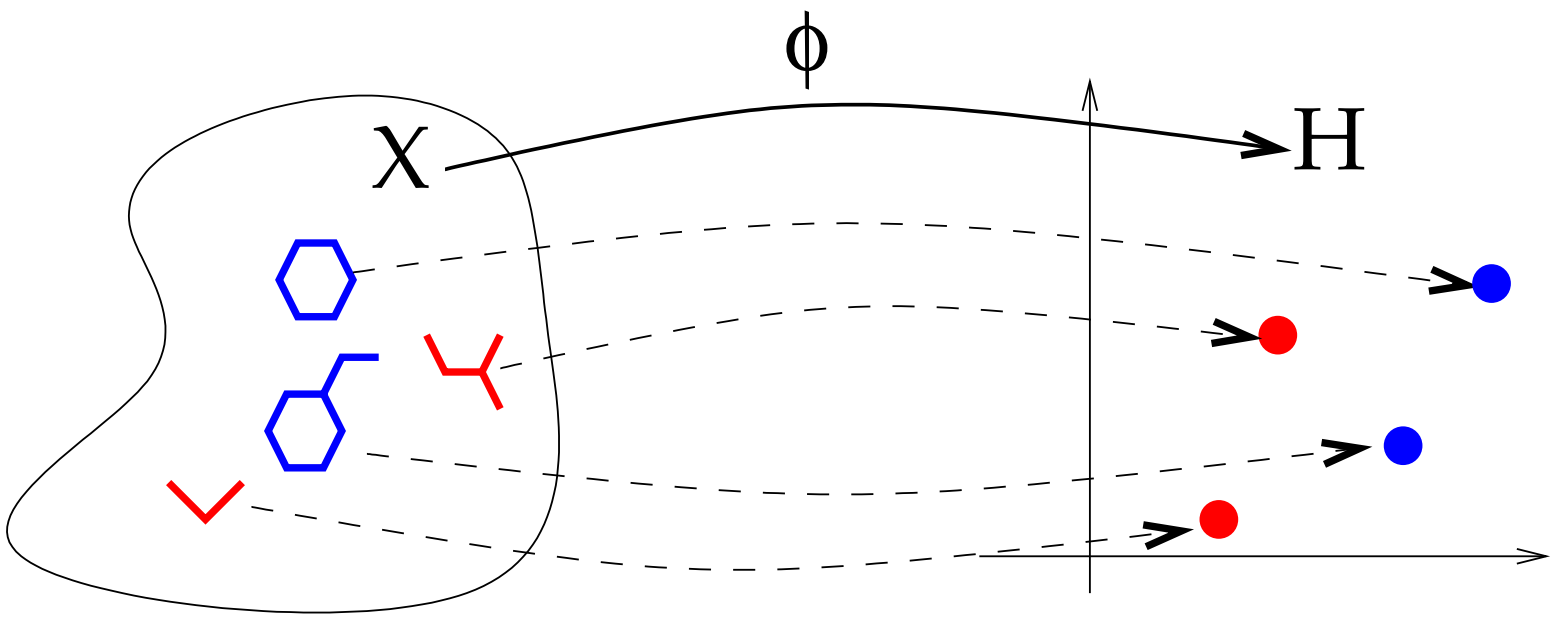


Classification with SVM

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 $\mathcal{H}$.

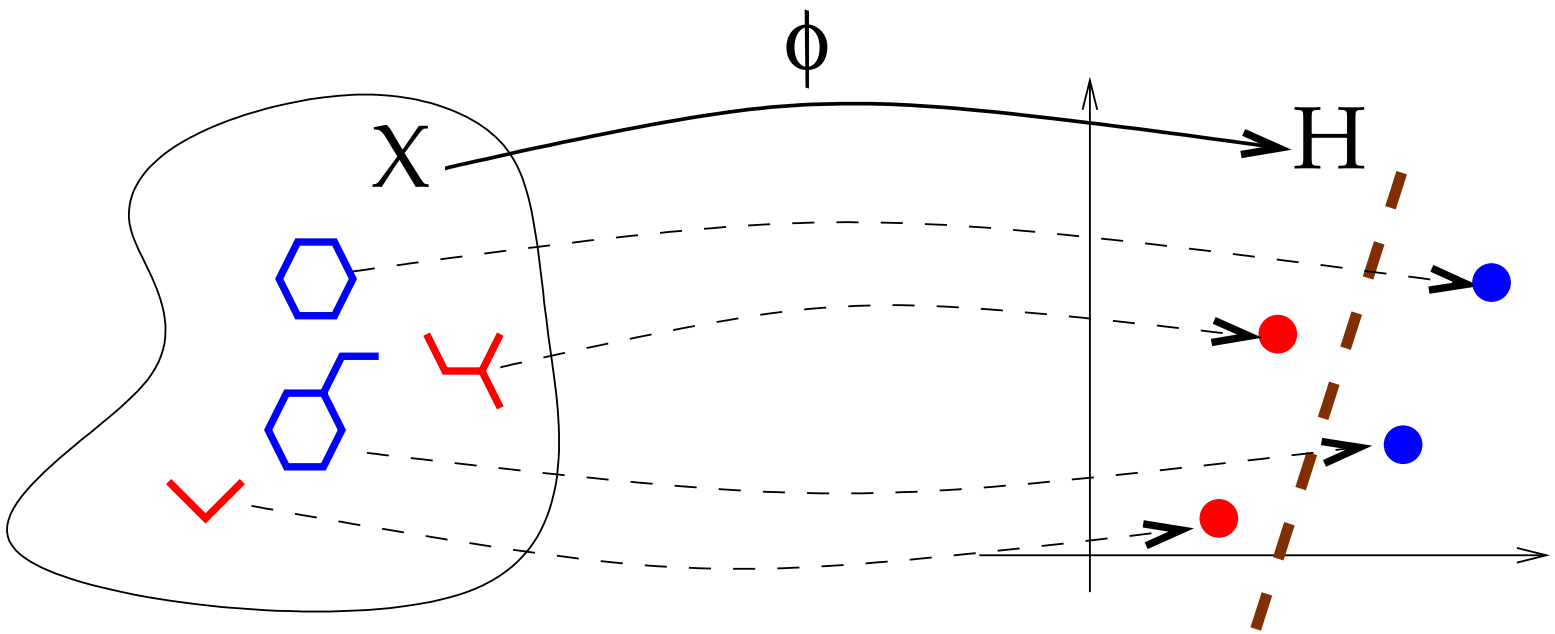


Classification with SVM

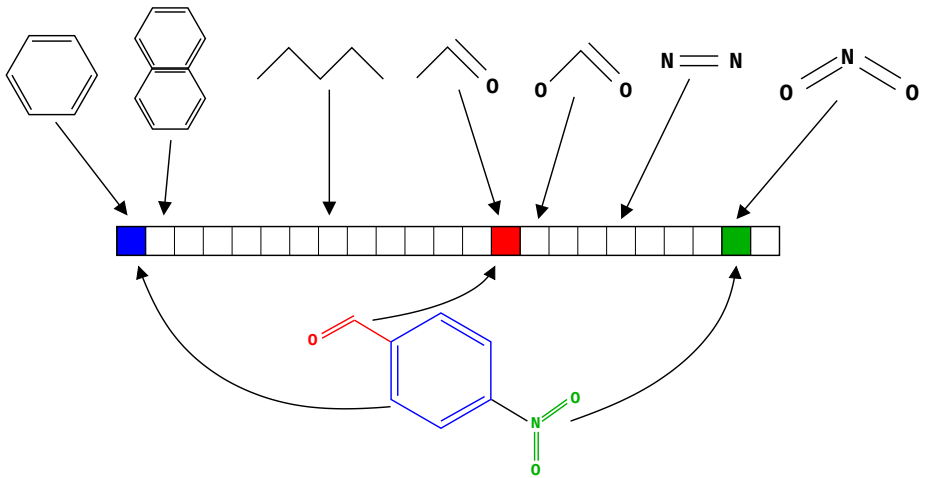
1. Represent each graph x by a vector $\Phi(x) \in \mathcal{H}$, either explicitly or implicitly through the kernel

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

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



Example: indexing by substructures

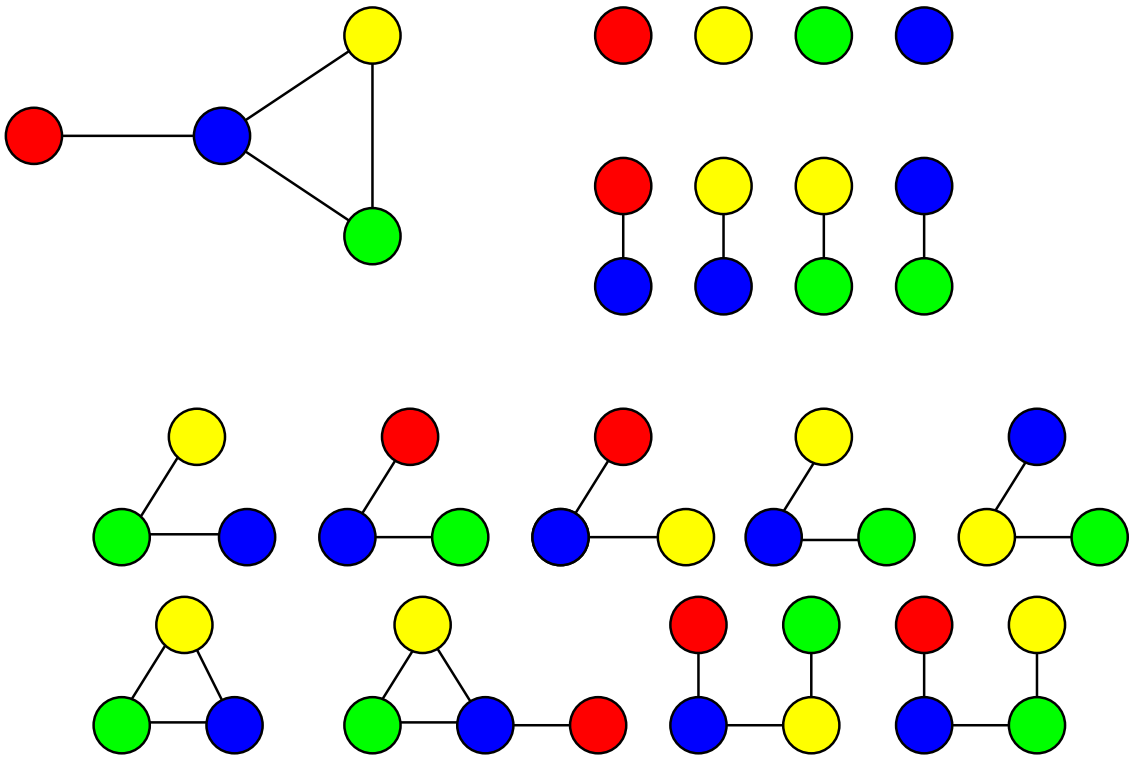


- Often we believe that the presence substructures are important predictive patterns
- Hence it makes sense to represent a graph by features that indicate the presence (or the number of occurrences) of particular substructures
- However, detecting the presence of particular substructures may be computationally challenging...

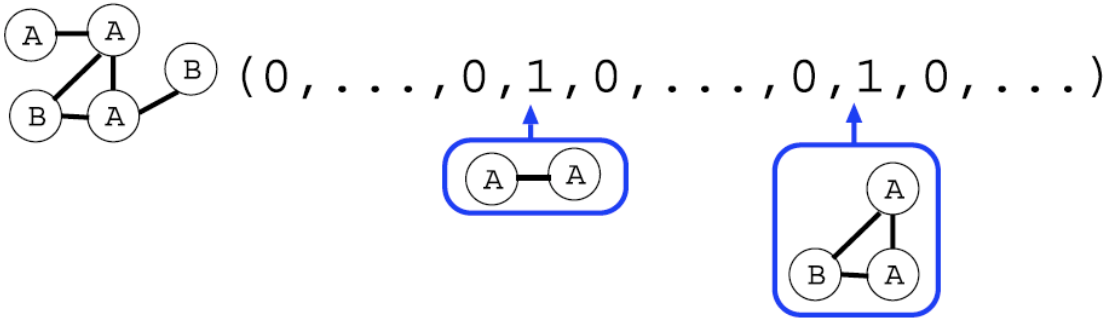
Subgraphs

Definition

A subgraph of a graph (V, E) is a connected graph (V', E') with $V' \subset V$ and $E' \subset E$.



Indexing by all subgraphs?



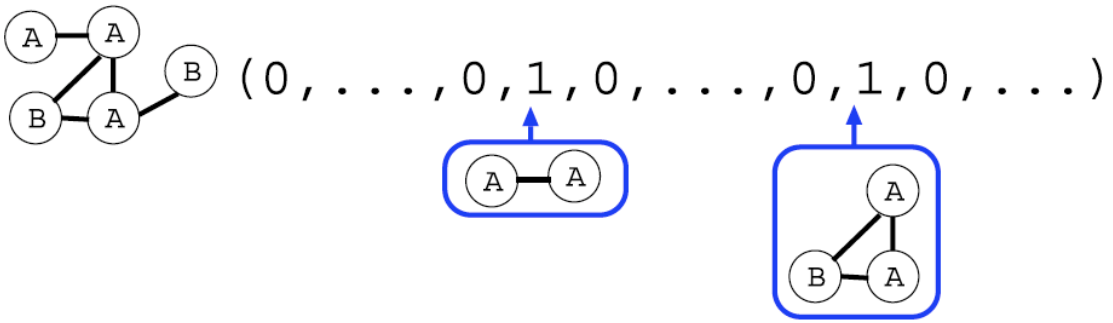
Theorem

1. *Computing all subgraph occurrences is NP-hard.*
2. *Computing the subgraph kernel is NP-hard.*

Proof.

1. Finding an occurrence of the linear path of size n is finding a Hamiltonian path, which is NP-complete.
2. Similarly, if we can compute the subgraph kernel then we can deduce the presence of a Hamiltonian path (left as exercice).

Indexing by all subgraphs?



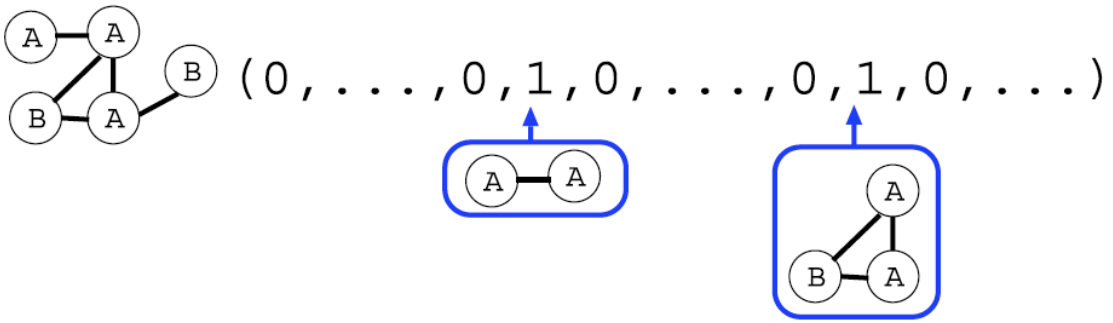
Theorem

1. *Computing all subgraph occurrences is NP-hard.*
2. *Computing the subgraph kernel is NP-hard.*

Proof.

1. Finding an occurrence of the linear path of size n is finding a Hamiltonian path, which is NP-complete.
2. Similarly, if we can compute the subgraph kernel then we can deduce the presence of a Hamiltonian path (left as exercice).

Indexing by all subgraphs?



Theorem

1. *Computing all subgraph occurrences is NP-hard.*
2. *Computing the subgraph kernel is NP-hard.*

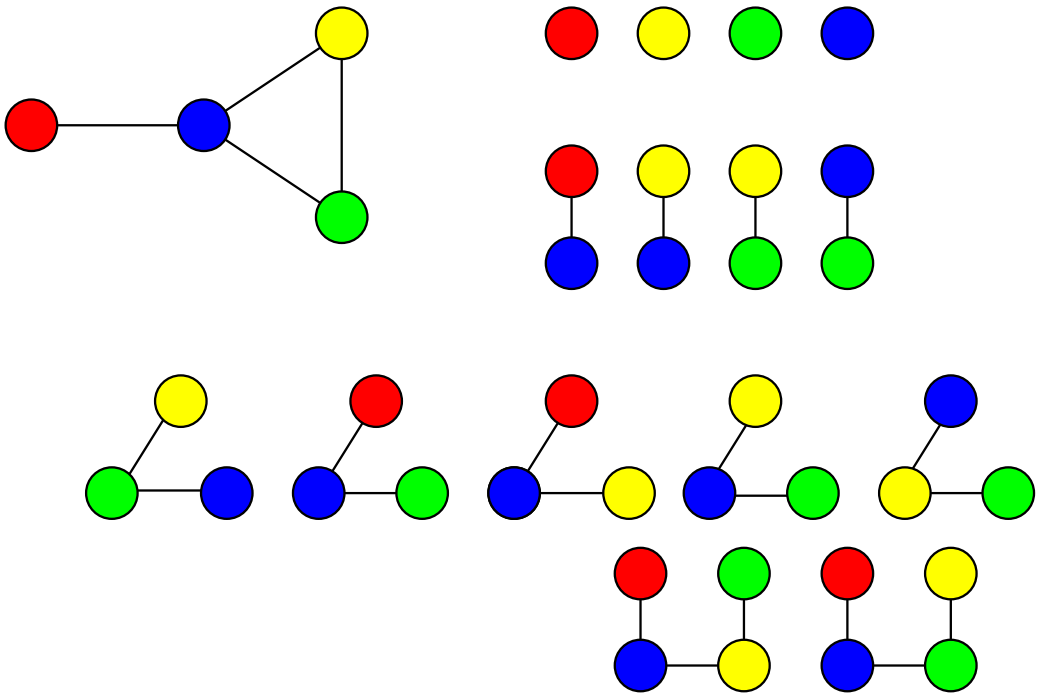
Proof.

1. Finding an occurrence of the linear path of size n is finding a Hamiltonian path, which is NP-complete.
2. Similarly, if we can compute the subgraph kernel then we can deduce the presence of a Hamiltonian path (left as exercice).

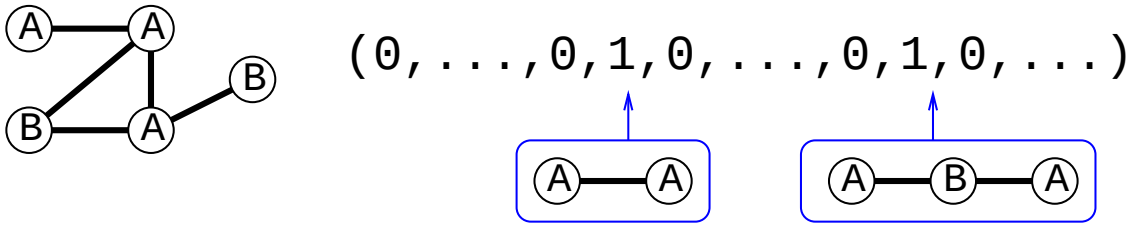
Paths

Definition

- A path of a graph (V, E) is sequence of distinct vertices $v_1, \ldots, v_n \in V$ $(i \neq j \implies v_i \neq v_j)$ such that $(v_i, v_{i+1}) \in E$ for $i = 1, \ldots, n-1$.
- Equivalently the paths are the linear subgraphs.



Indexing by all paths?



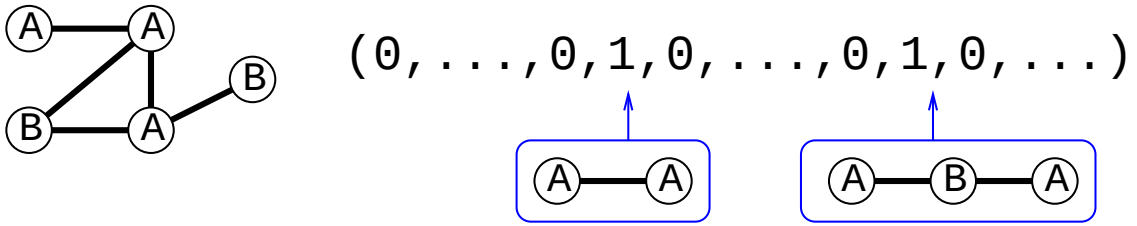
Theorem

1. *Computing all path occurrences is NP-hard.*
2. *Computing the path kernel is NP-hard*

Proof.

Same as for subgraphs. □

Indexing by all paths?



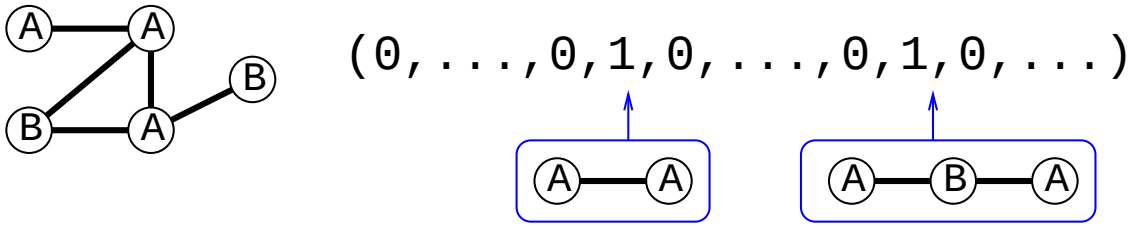
Theorem

1. *Computing all path occurrences is NP-hard.*
2. *Computing the path kernel is NP-hard*

Proof.

Same as for subgraphs. □

Indexing by all paths?



Theorem

1. *Computing all path occurrences is NP-hard.*
2. *Computing the path kernel is NP-hard*

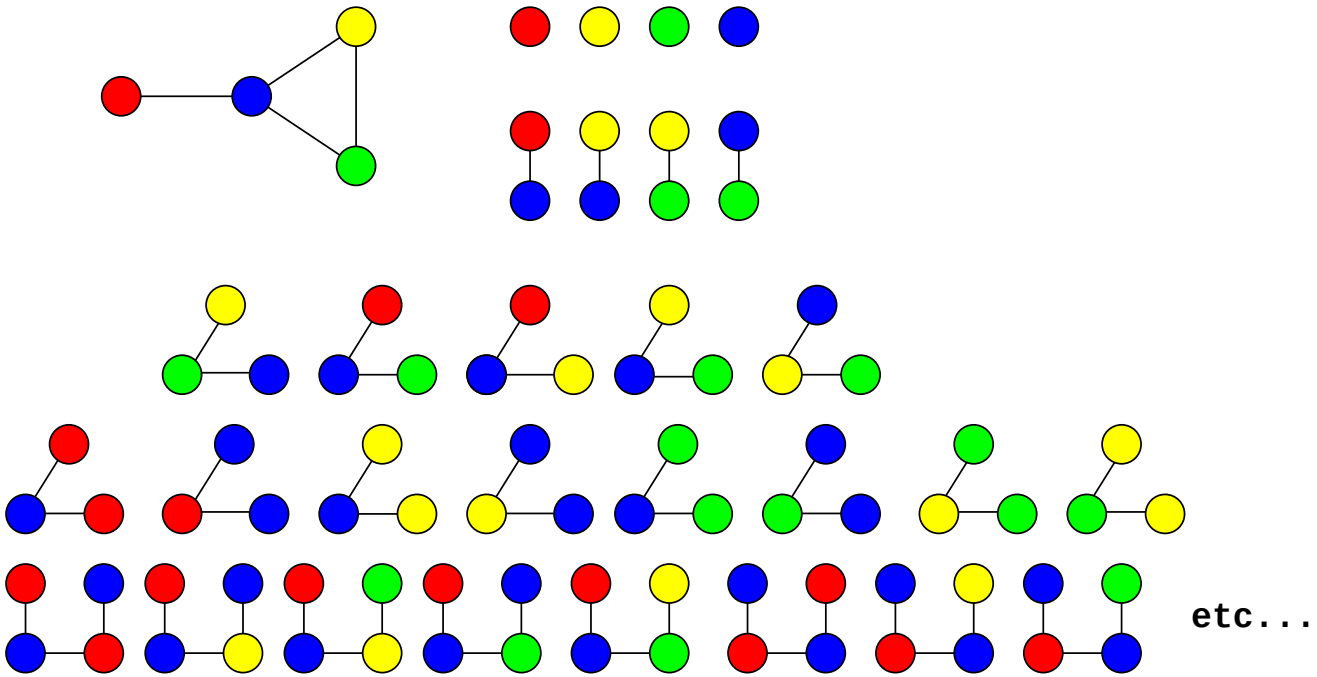
Proof.

Same as for subgraphs. □

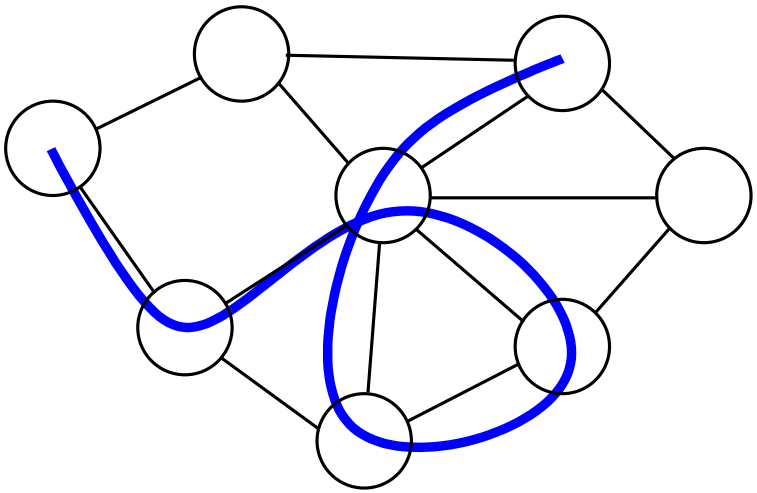
Walks

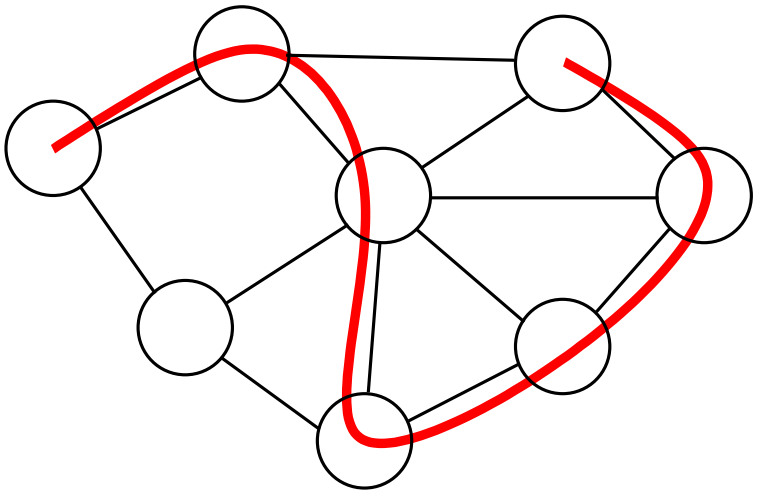
Definition

- 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

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

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

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

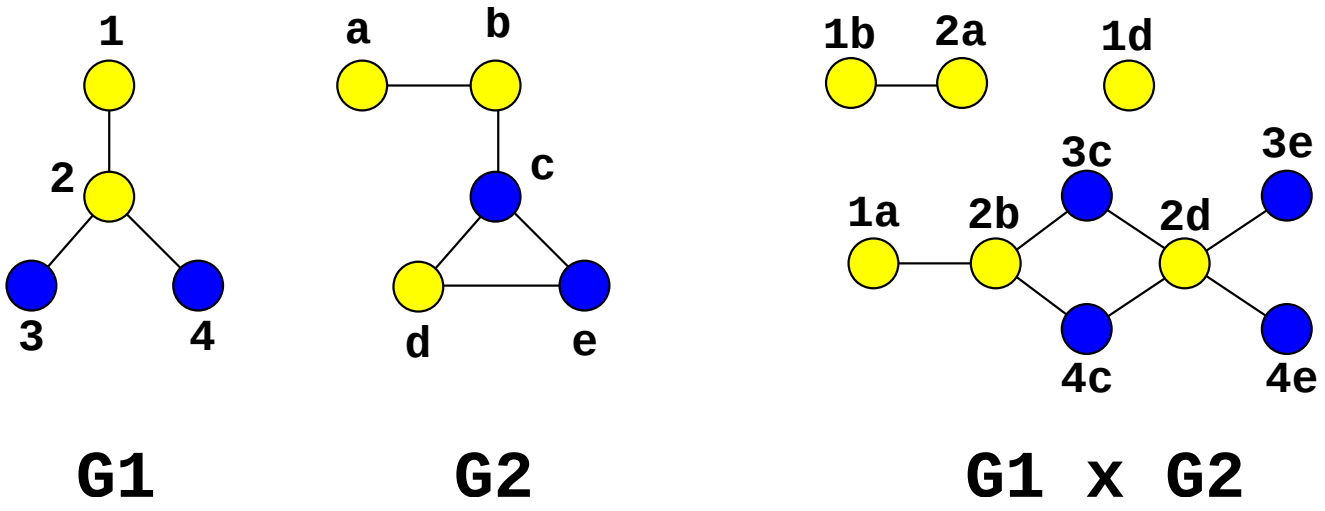
These three kernels (*n*th-order, random and geometric walk kernels) can be computed efficiently in polynomial time.

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



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:

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

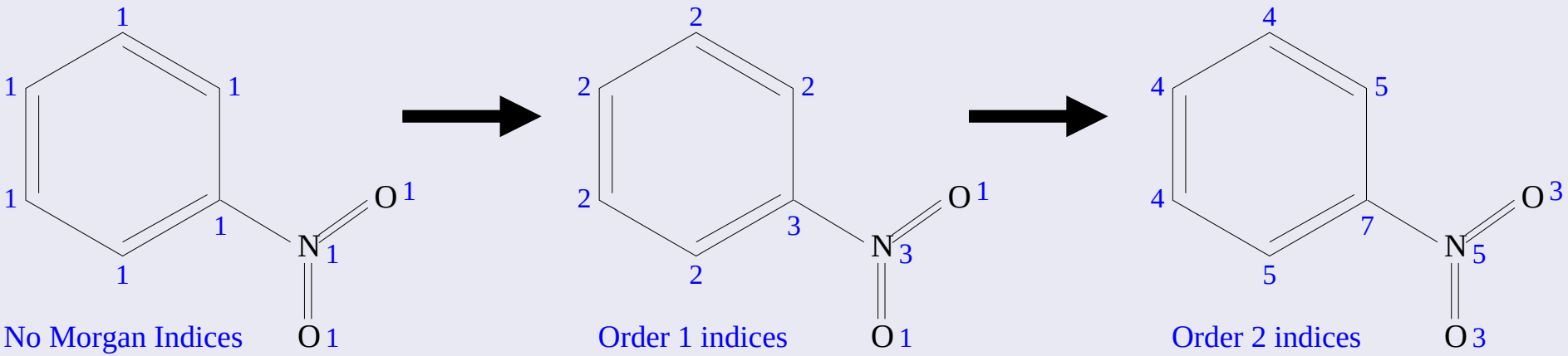
- Let Λ_i be the vector of $\lambda^i(v)$ and Λ_t be the matrix of $\lambda^t(v, v')$:

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

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

Extensions 1: label enrichment

Atom relabebling with the Morgan index

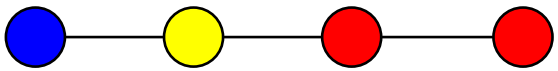


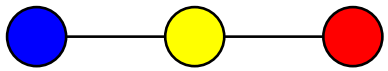
- Compromise between fingerprints and structural keys features.
- Other relabeling schemes are possible (graph coloring).
- Faster computation with more labels (less matches implies a smaller product graph).

Extension 2: Non-tottering walk kernel

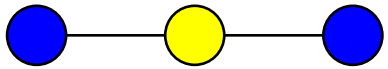
Tottering walks

A tottering walk is a walk $w = v_1 \dots v_n$ with $v_i = v_{i+2}$ for some i.





Non-tottering

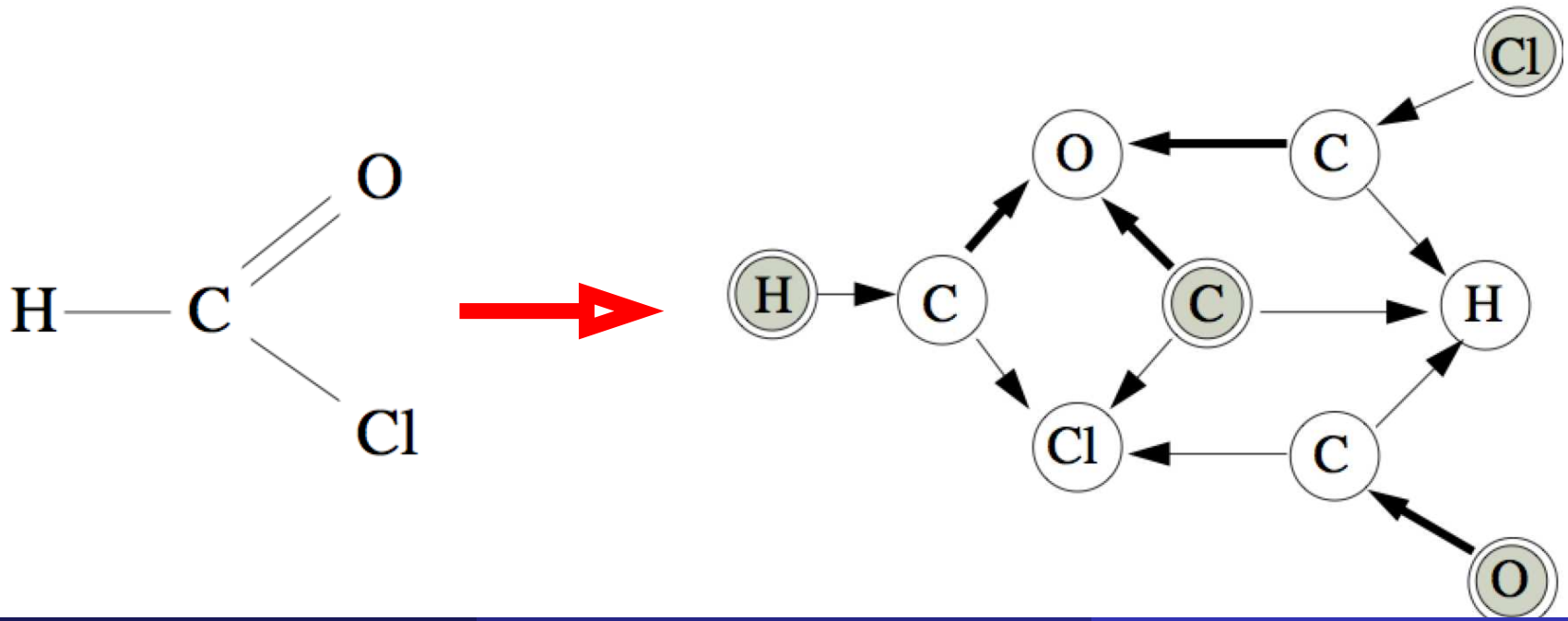


Tottering

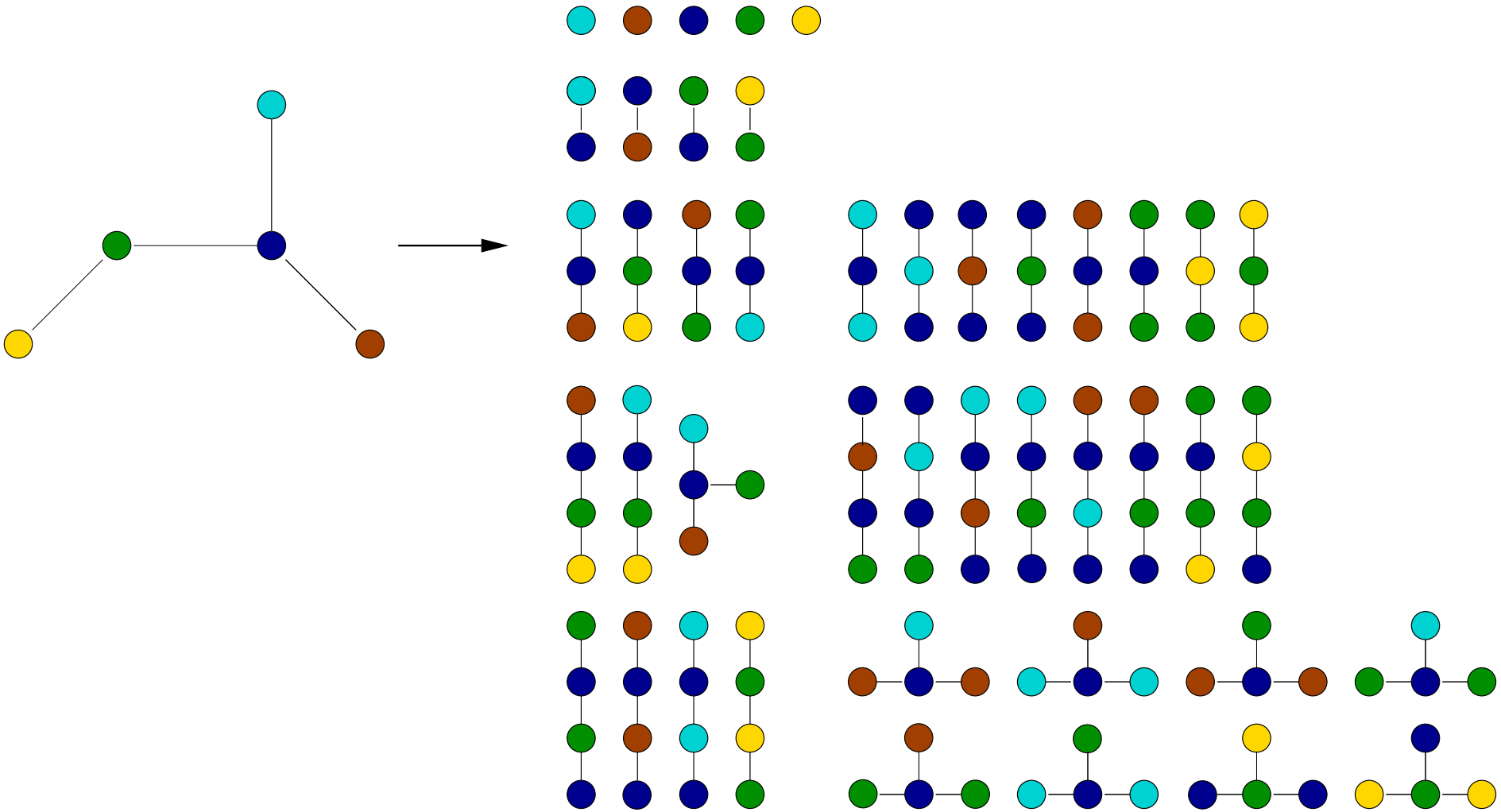
- Tottering walks seem irrelevant for many applications
- Focusing on non-tottering walks is a way to get closer to the path kernel (e.g., equivalent on trees).

Computation of the non-tottering walk kernel (Mahé et al., 2005)

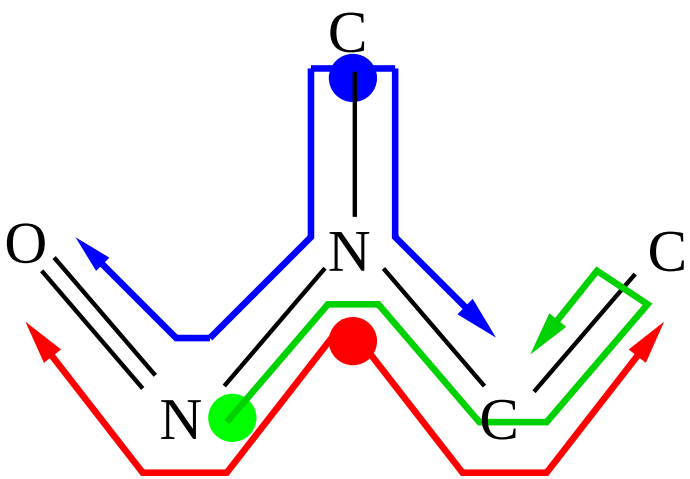
- Second-order Markov random walk to prevent tottering walks
- Written as a first-order Markov random walk on an augmented graph
- Normal walk kernel on the augmented graph (which is always a directed graph).

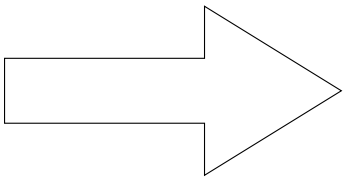


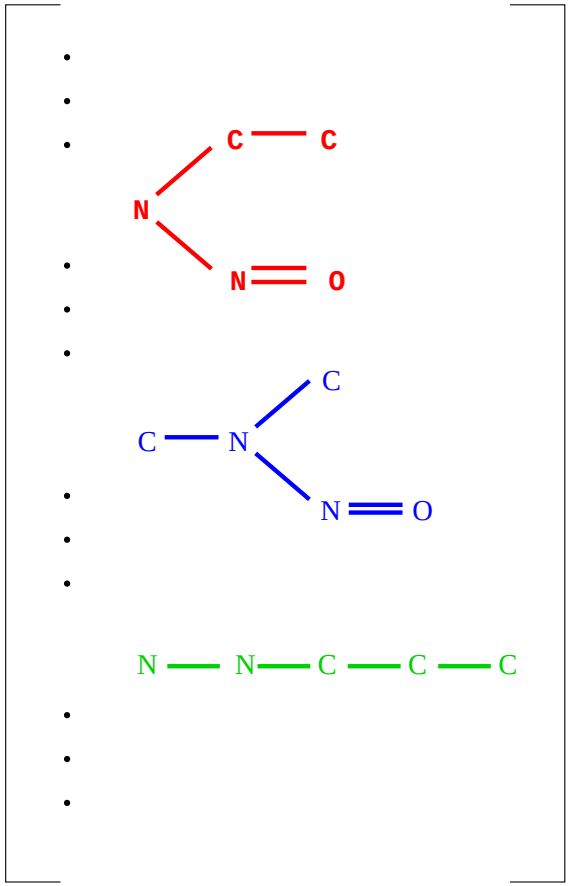
Extension 3: Subtree kernels



Example: Tree-like fragments of molecules







Computation of the subtree kernel

- Like the walk kernel, amounts to compute the (weighted) number of subtrees in the product graph.
- Recursion: if $\mathcal{T}(v, n)$ denotes the weighted number of subtrees of depth n rooted at the vertex v, then:

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

  where $\mathcal{N}(v)$ is the set of neighbors of v.
- Can be combined with the non-tottering graph transformation as preprocessing to obtain the non-tottering subtree kernel.

Application in chemoinformatics (Mahé et al., 2004)

MUTAG dataset

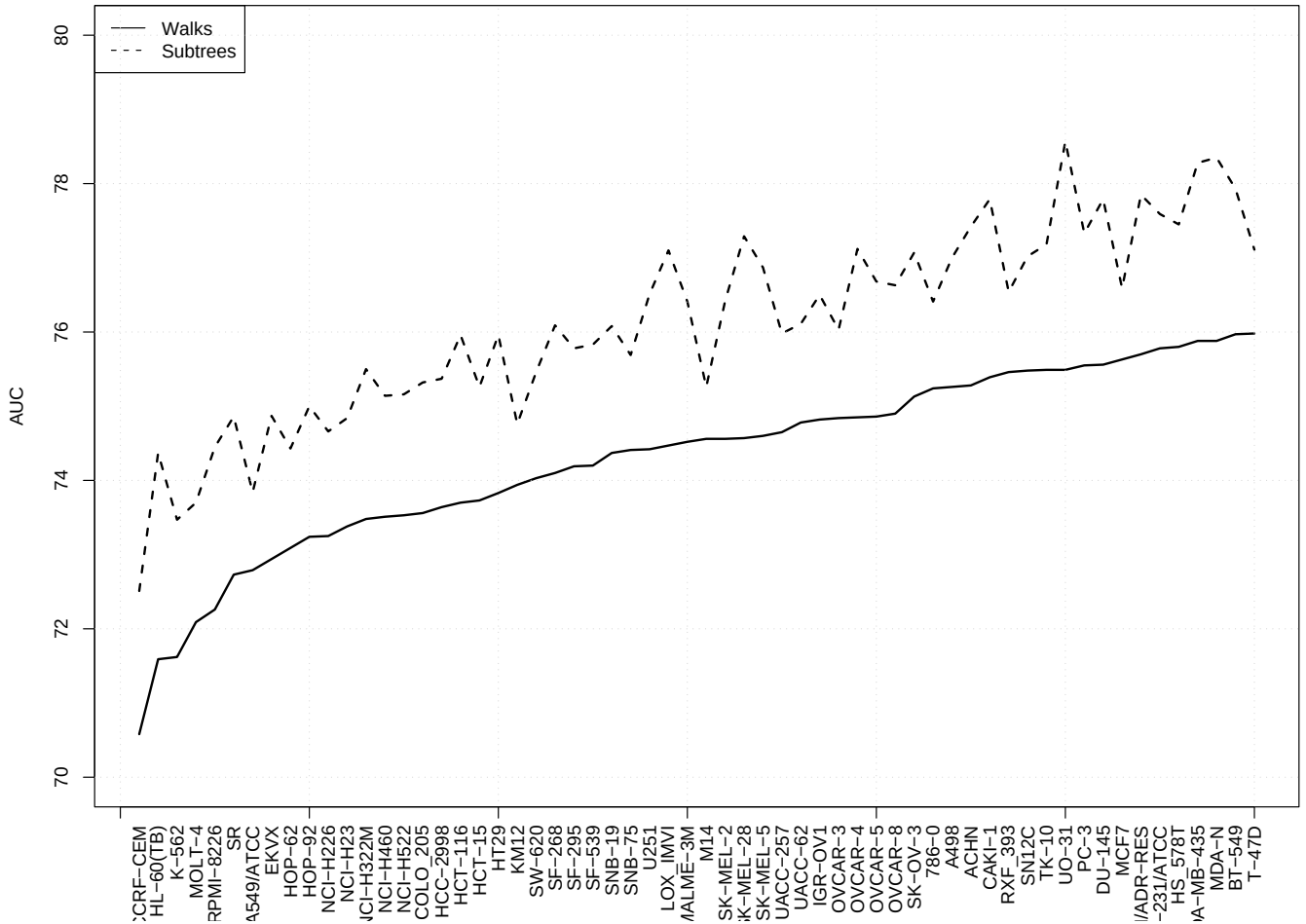
- aromatic/hetero-aromatic compounds
- high mutagenic activity /no mutagenic activity, assayed in *Salmonella typhimurium*.
- 188 compouunds: 125 + / 63 -

Results

10-fold cross-validation accuracy

Method	Accuracy
Progol1	81.4%
2D kernel	91.2%

2D Subtree vs walk kernels (Mahé and V., 2009)



Screening of inhibitors for 60 cancer cell lines.

Summary: graph kernels

What we saw

- Kernels do not allow to overcome the NP-hardness of subgraph patterns
- They allow to work with approximate subgraphs (walks, subtrees), in infinite dimension, thanks to the kernel trick
- They give state-of-the-art results

Outline

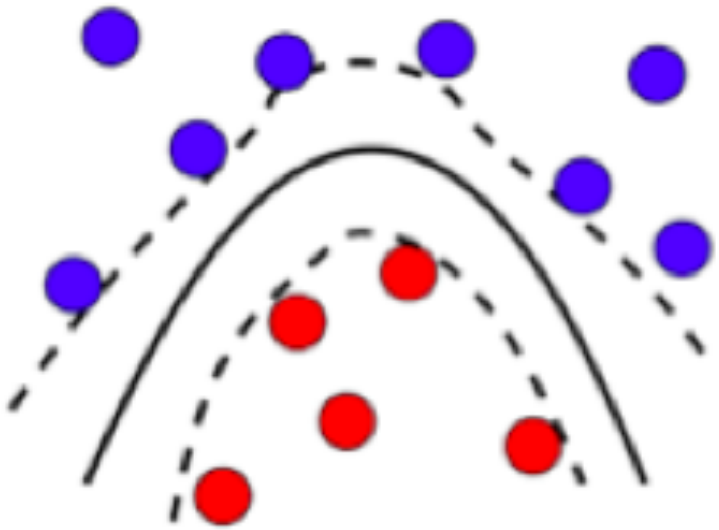
1. Machine learning in bioinformatics
2. Linear support vector machines
3. Nonlinear SVM and kernels
4. SVM for complex data: the case of graphs
5. Conclusion

Machine learning in computational biology

- Biology faces a flood of data following the development of high-throughput technologies (sequencing, DNA chips, ...)
- Many problems can be formalized in the framework of machine learning, e.g.:
  - Diagnosis, prognosis
  - Protein annotation
  - Drug discovery, virtual screening
- These data have often complex structures (strings, graphs, high-dimensional vectors) and often require dedicated algorithms.

Support vector machines (SVM)

- A general-purpose algorithm for pattern recognition
- Based on the principle of large margin *("séparateur à vaste marge")*
- Linear or nonlinear with the kernel trick
- Control of the regularization / data fitting trade-off with the C parameter
- State-of-the-art performance on many applications



Kernels

- A central ingredient of SVM
- Allows nonlinearity
- Allows to work implicitly in a high-dimensional feature space
- Allows to work with structured data (e.g., graphs)