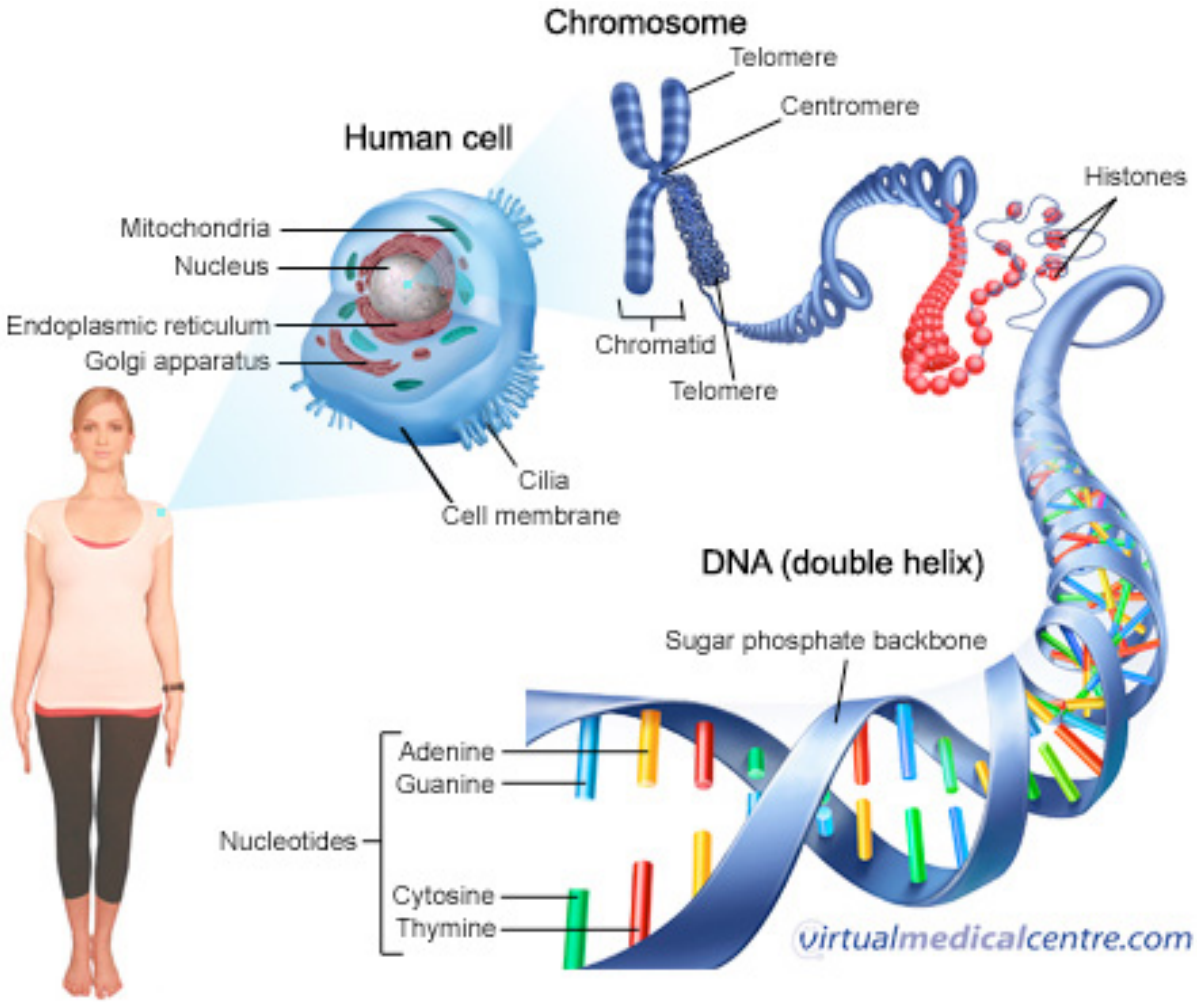
Machine learning for computational biology

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

Outline

Outline

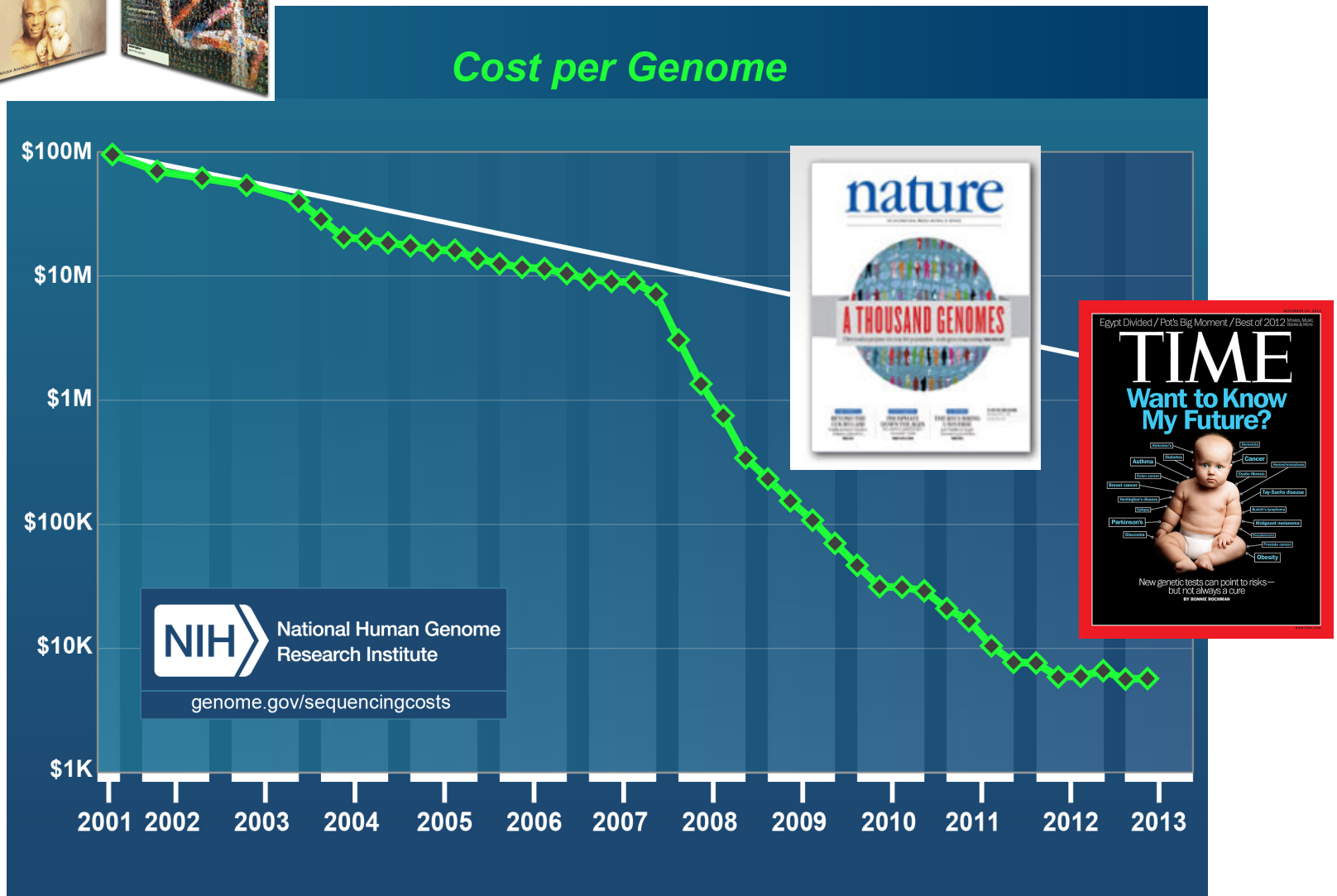
What's in your body



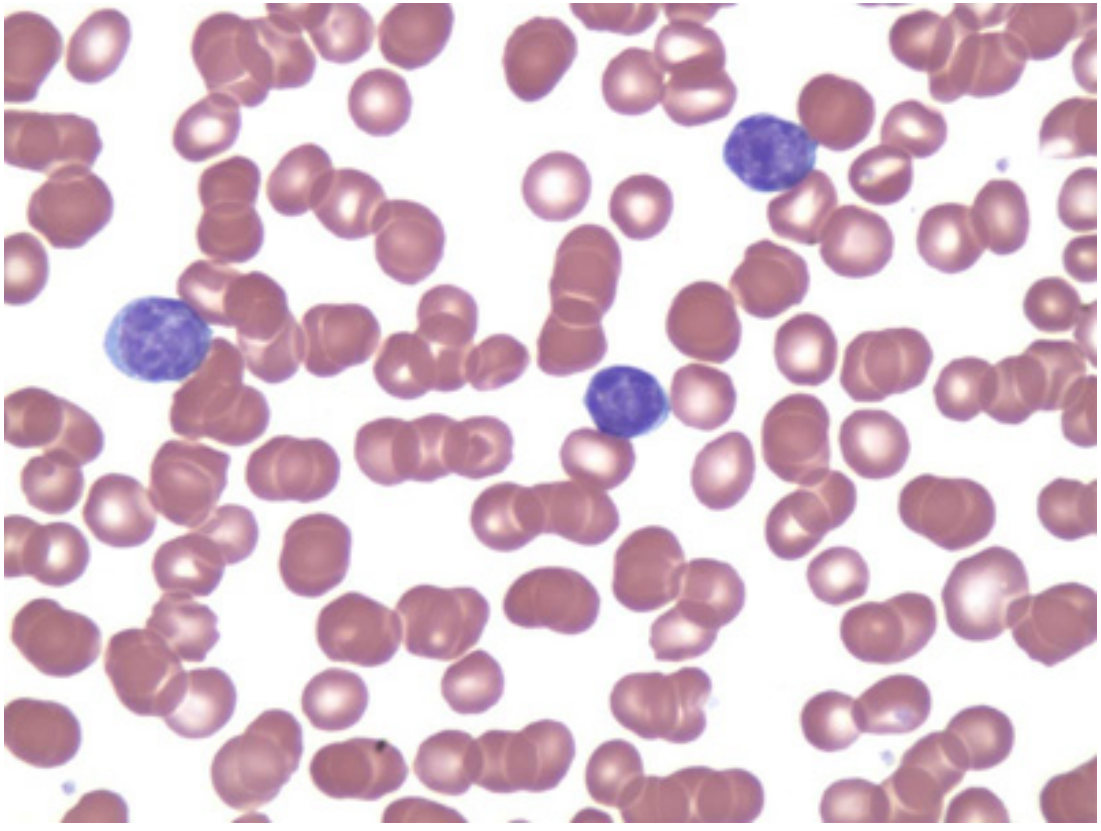
1 body = 10^{14} human cells (and 100x more non-human cells)

1 cell = 6×10^9 ACGT coding for $20,000$ genes

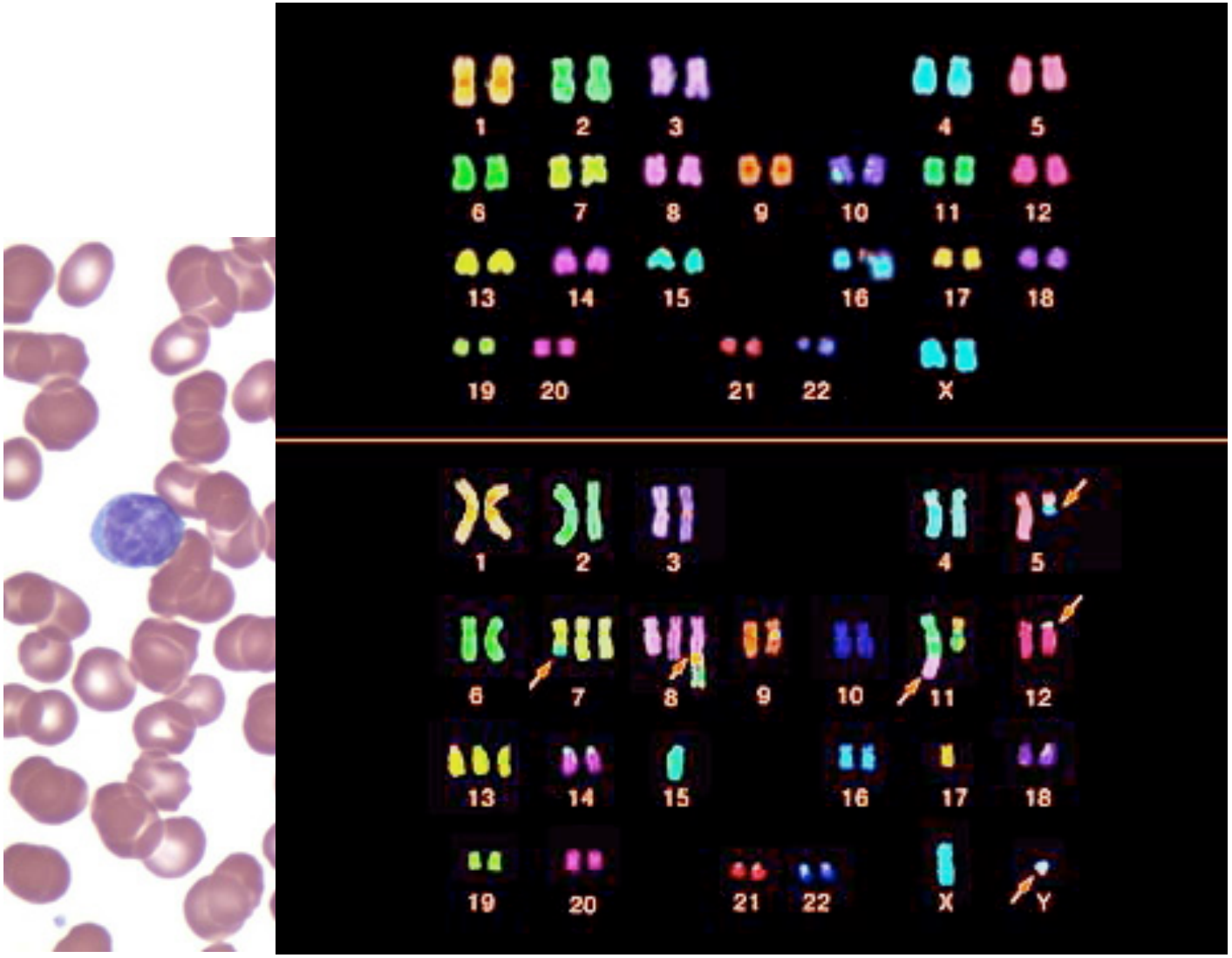
Sequencing revolution



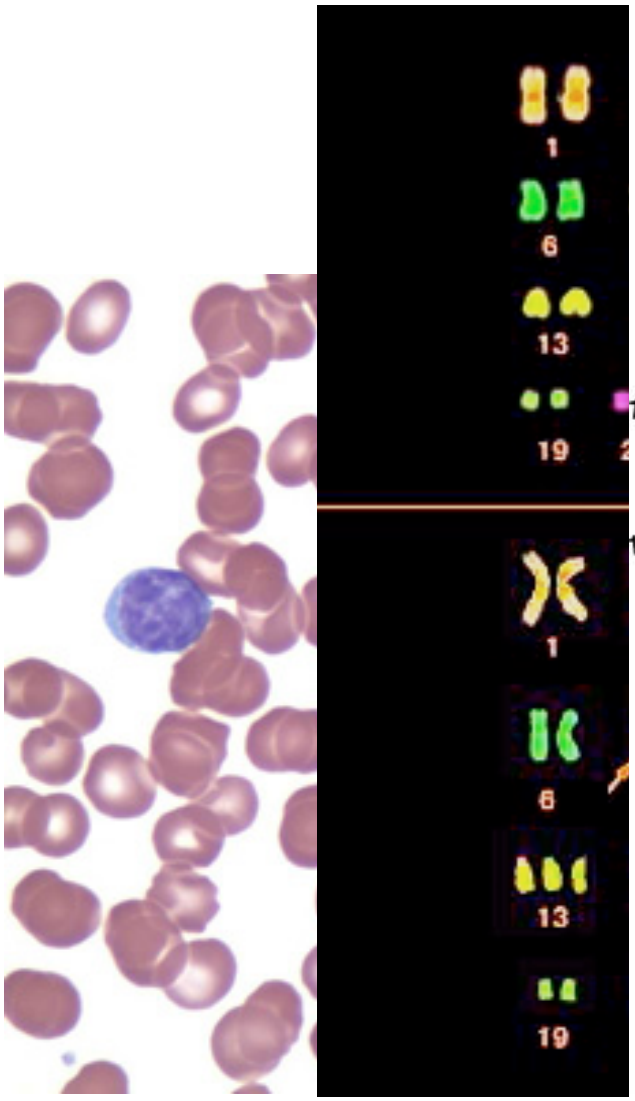
A cancer cell

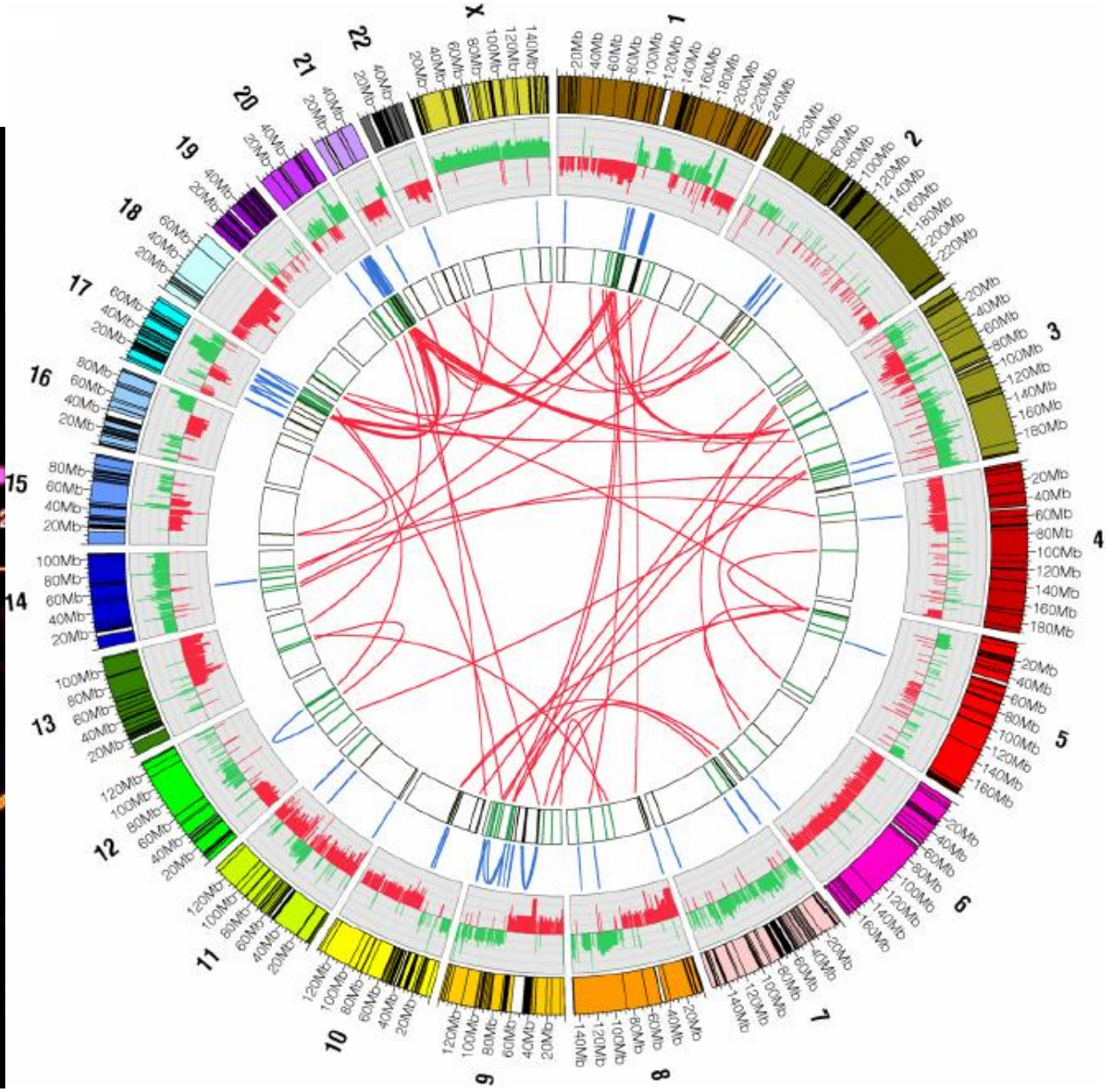


A cancer cell

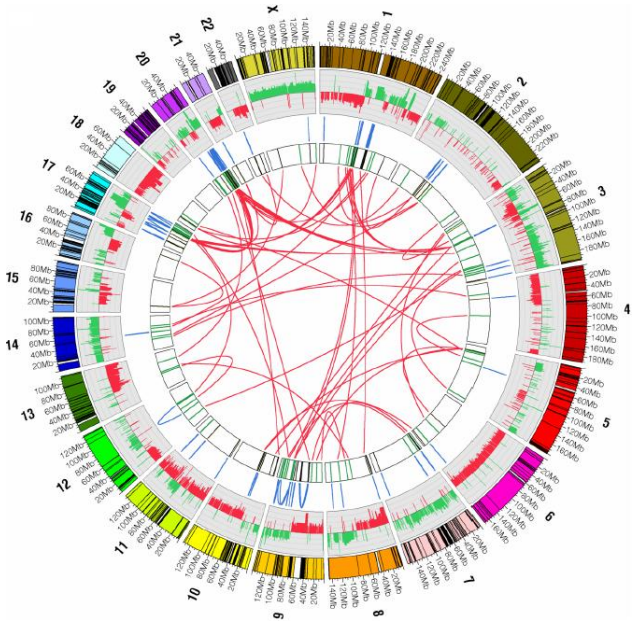


A cancer cell



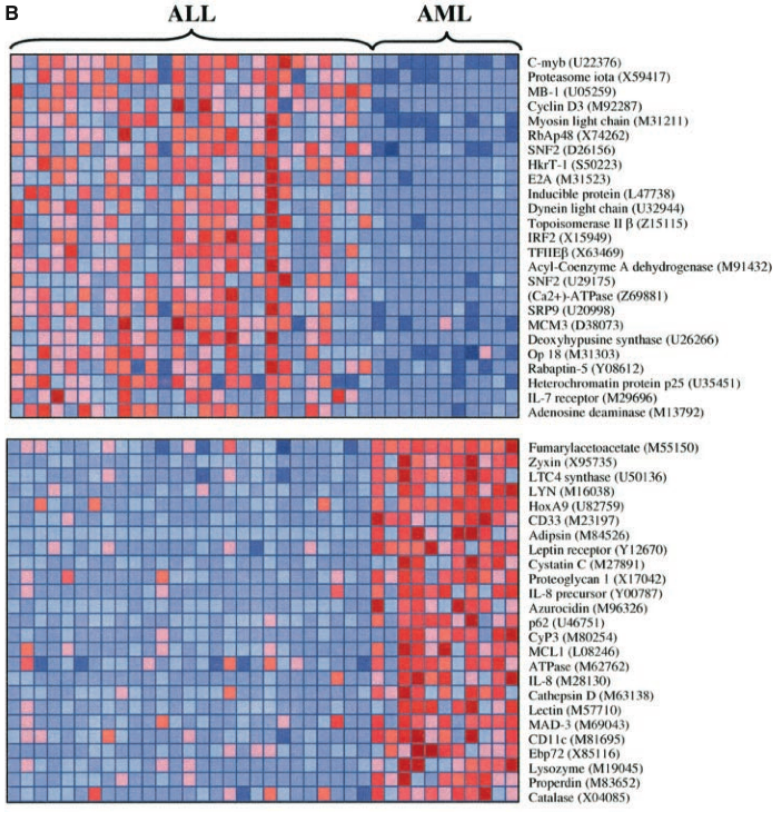


Opportunities



- What is your risk of developing a cancer? (*prevention*)
- After diagnosis and treatment, what is the risk of relapse? (*prognosis*)
- What specific treatment will cure your cancer? (*personalized medicine*)

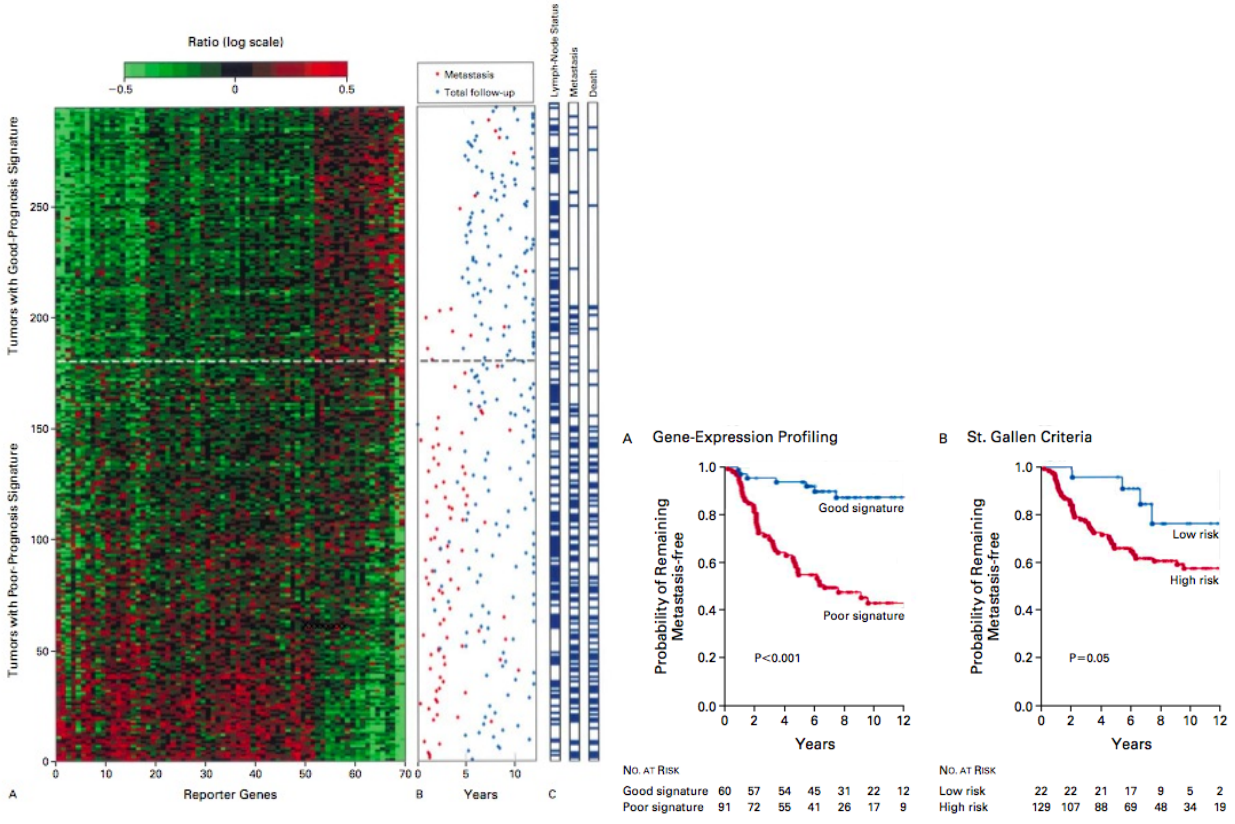
Cancer diagnosis



Problem 1

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

Cancer prognosis



Problem 2

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

Pharmacogenomics / Toxicogenomics

Problem 3

Given the genome of a person, which drug should we give?

Protein annotation

Data available

- Secreted proteins:

  MASKATLLLAFTLLFATCIARHQQRQQQQNQCQLQNIEA...

  MARSSLFTFLCLAVFINGCLSQIEQQSPWEFQGSEVW...

  MALHTVLIMLSLLPMLEAQNPEHANITIGEPITNETLGWL...

  ...

- Non-secreted proteins:

  MAPPSVFAEVPQAQPVLVFKLIADFREDPDPRKVNLGVG...

  MAHTLGLTQPNSTEPHKISFTAKEIDVIEWKGDILVVG...

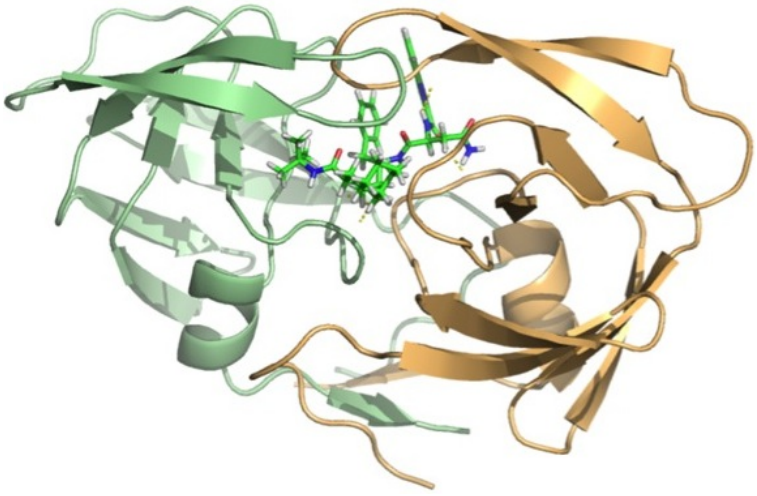
  MSISESYAKEIKTAFRQFTDFPIEGEQFEDFLPIIGNP..

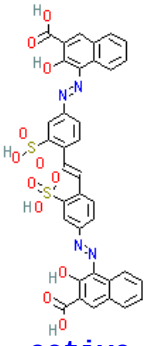
  ...

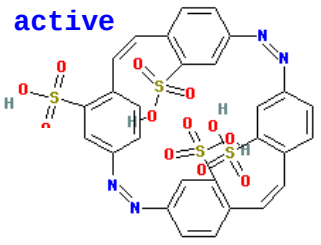
Problem 4

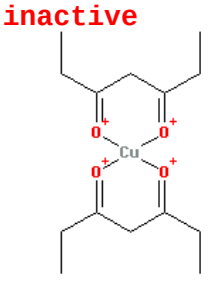
Given a newly sequenced protein, is it secreted or not?

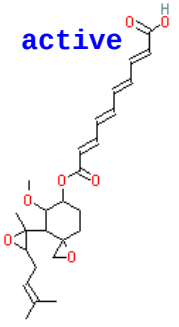
Drug discovery

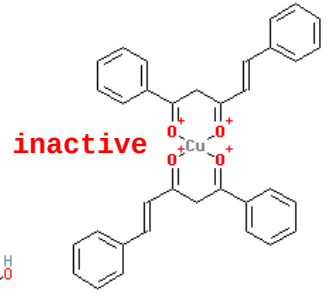


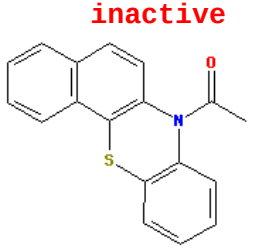








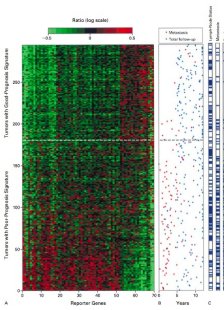


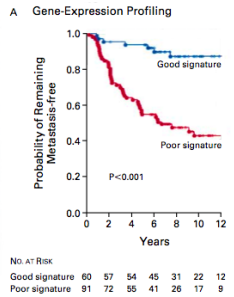


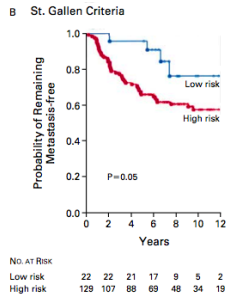
Problem 5

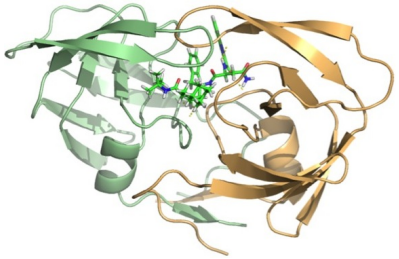
Given a new candidate molecule, is it likely to be active?

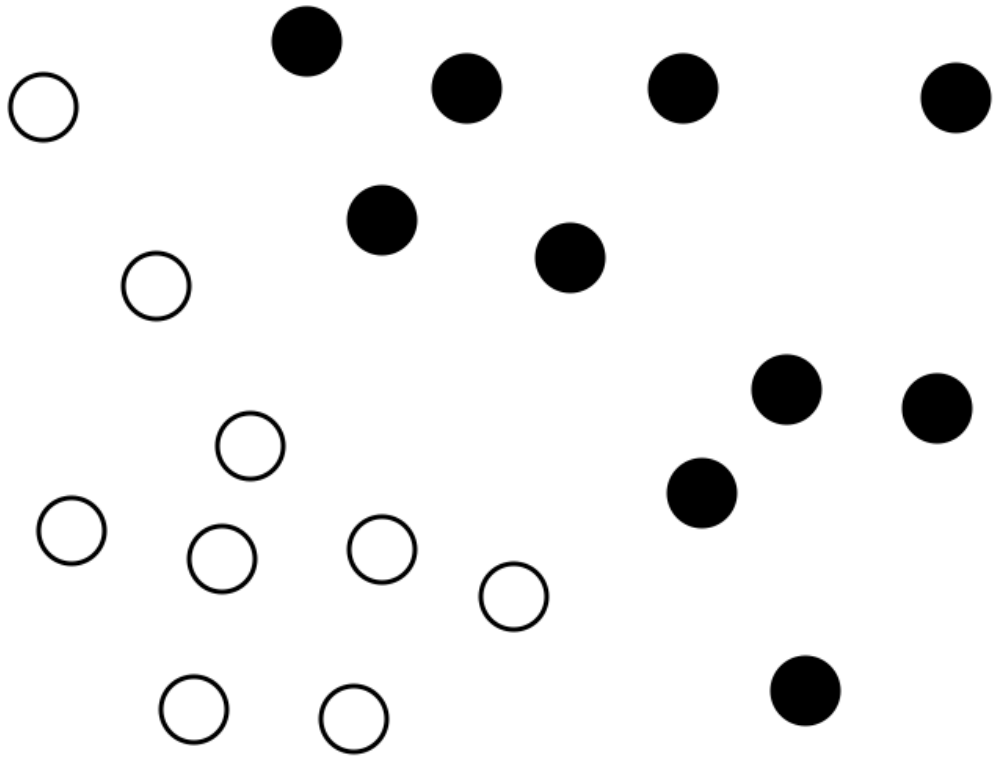
A common topic



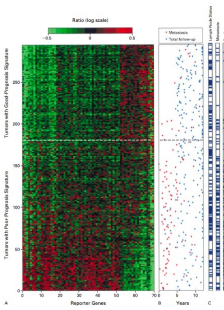


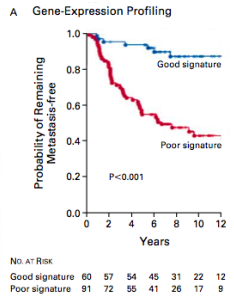


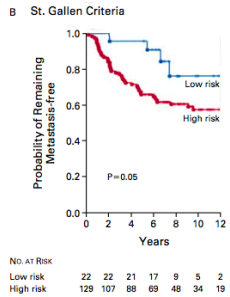


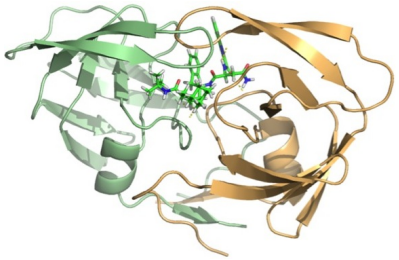


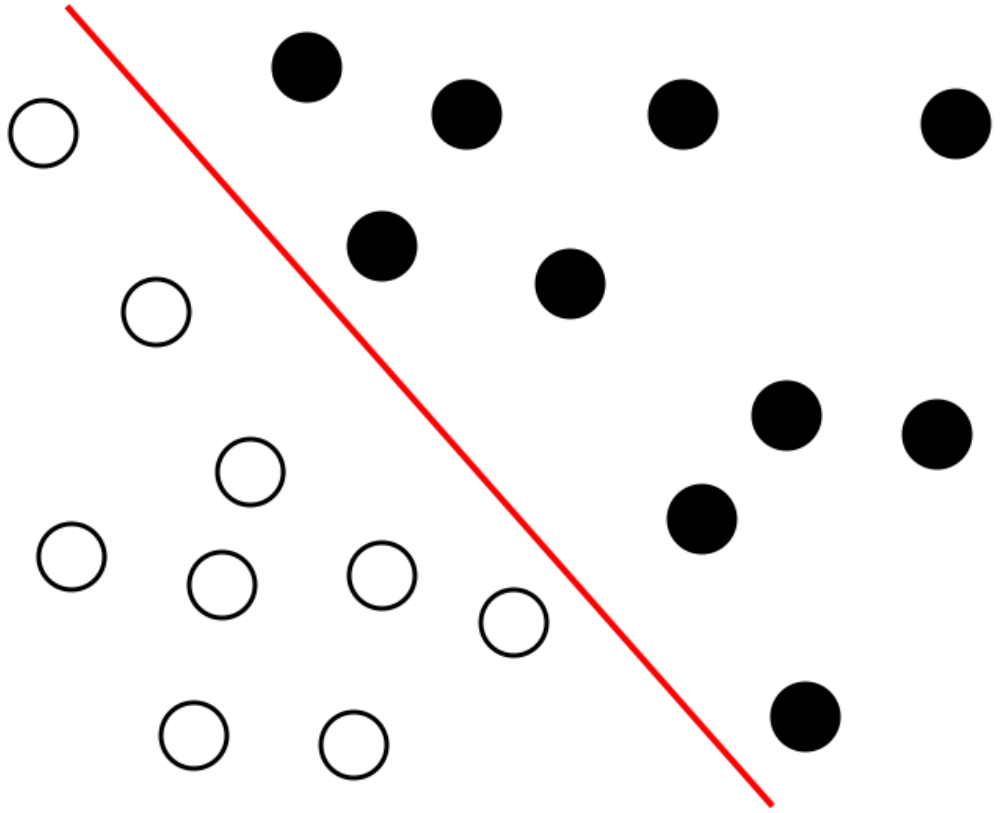
A common topic



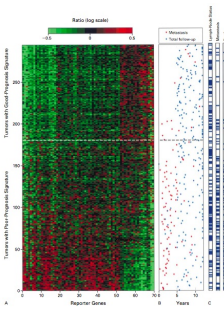


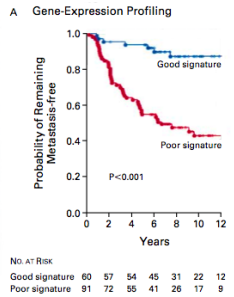


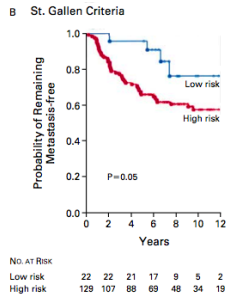


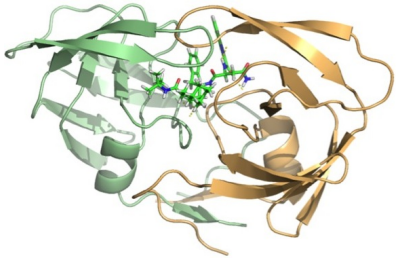


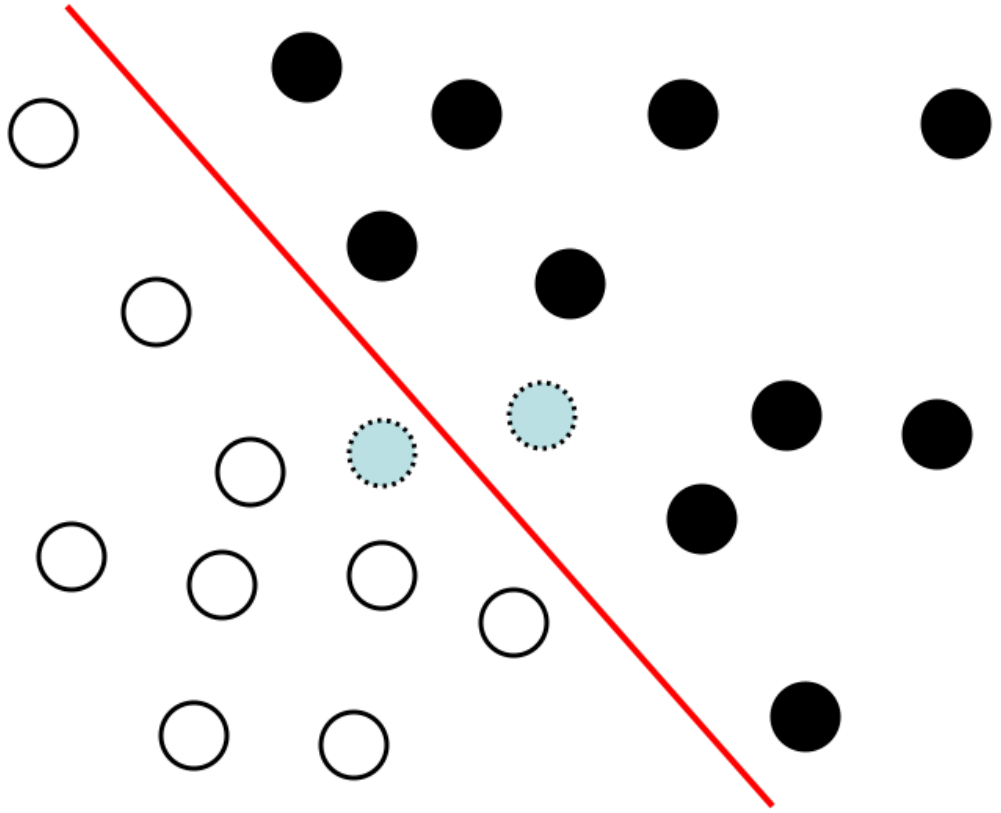
A common topic



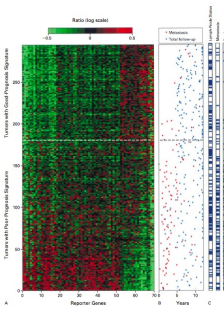


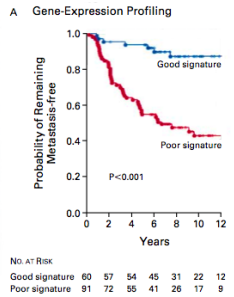


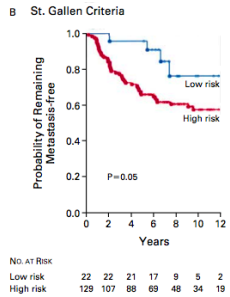


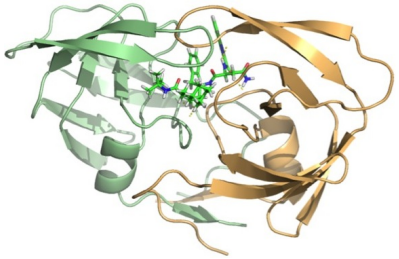


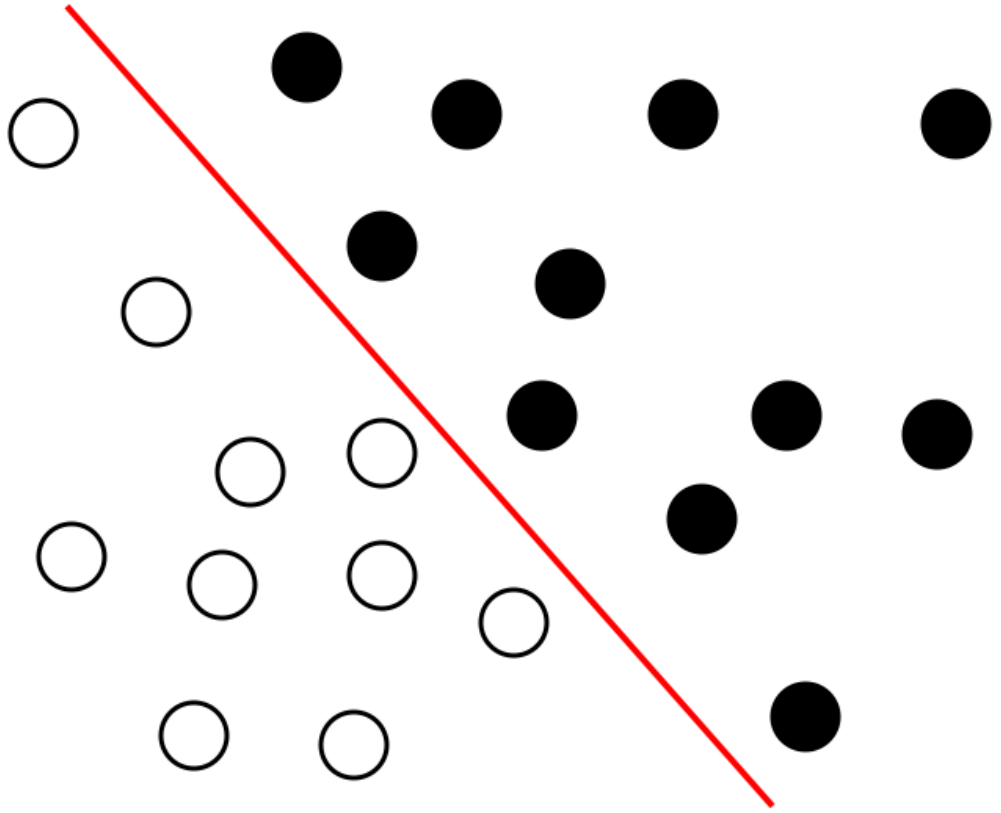
A common topic



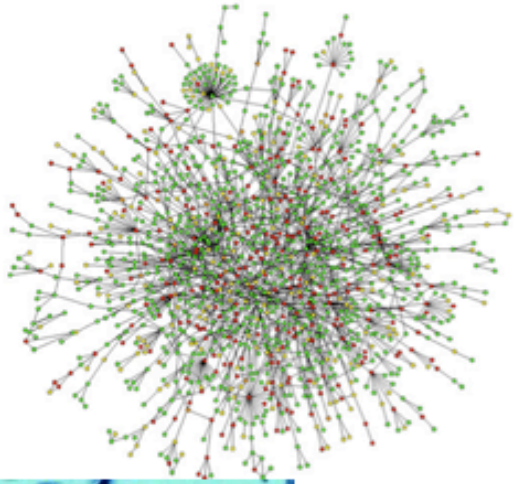


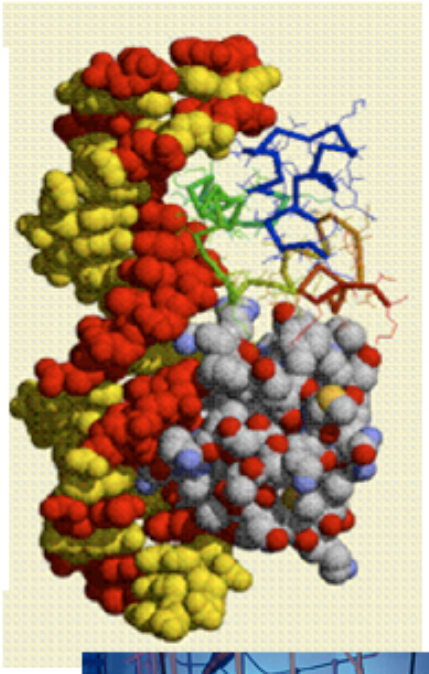


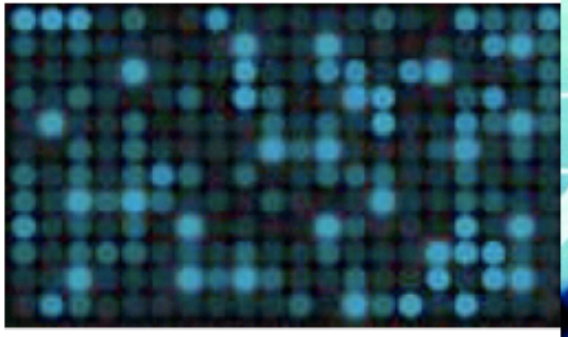


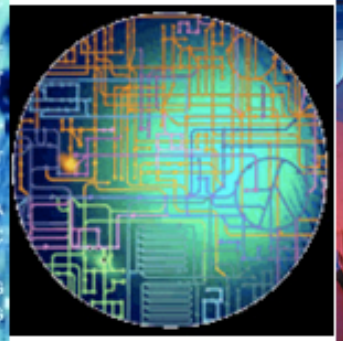


On real data...

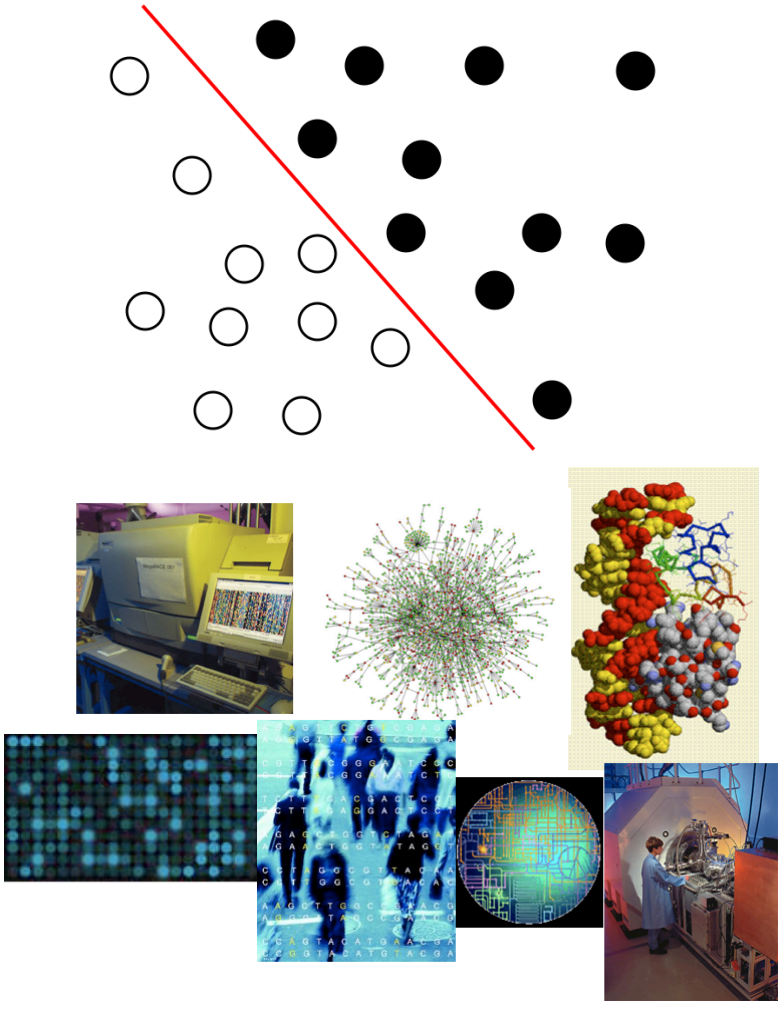








Pattern recognition, *aka* supervised classification

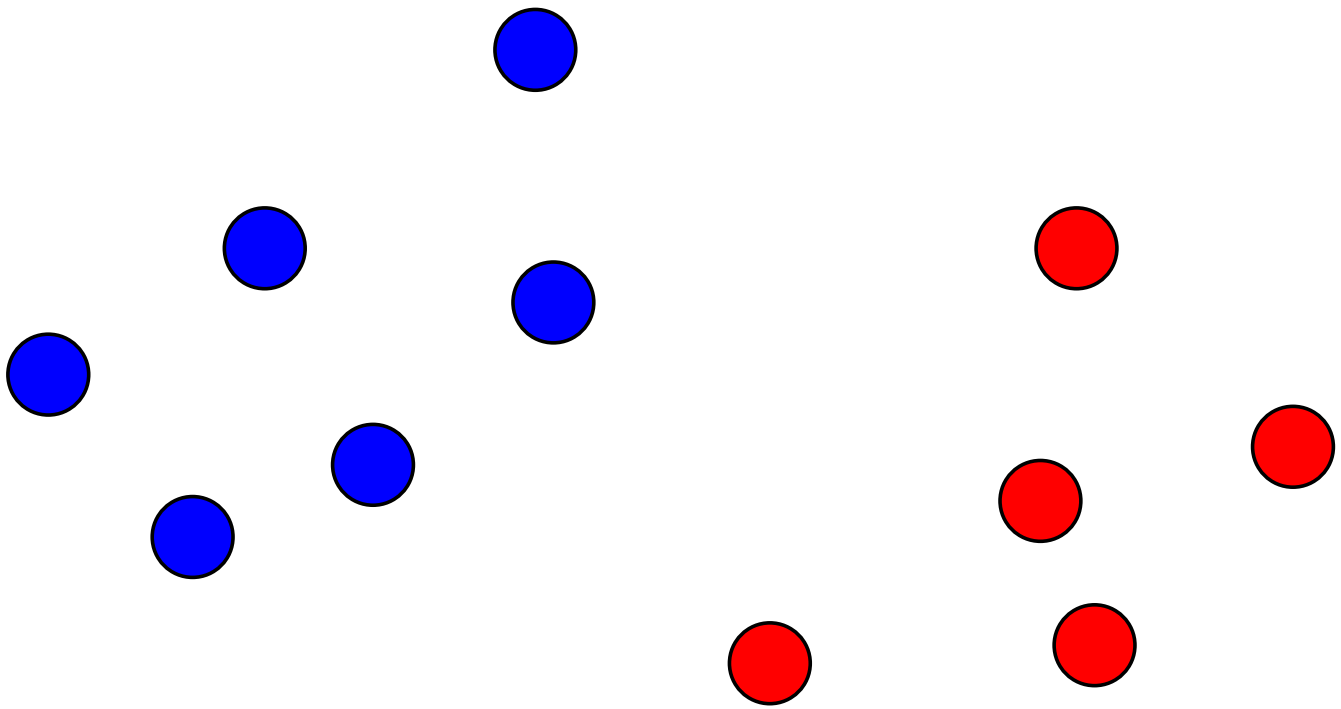


Challenges

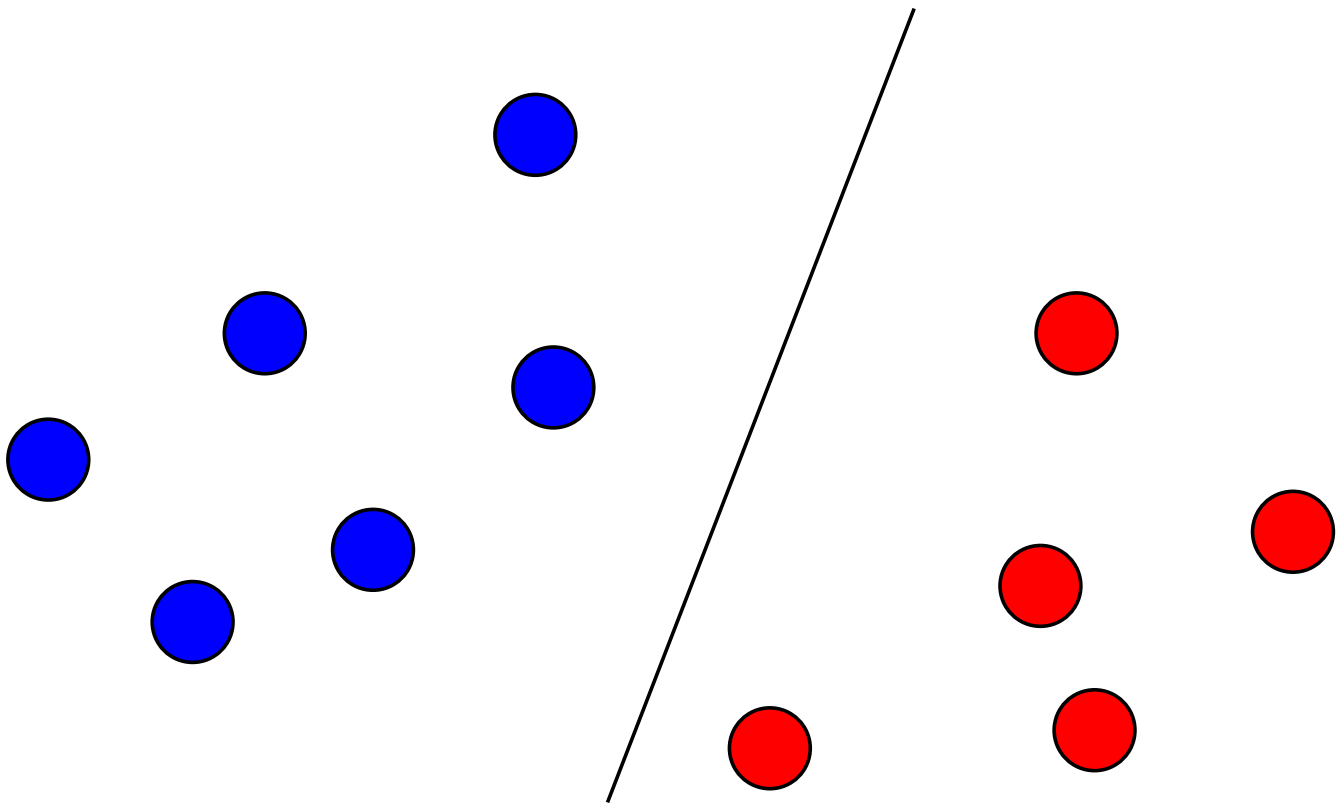
- High dimension
- Few samples
- Structured data
- Heterogeneous data
- Prior knowledge
- Fast and scalable implementations
- Interpretable models

Outline

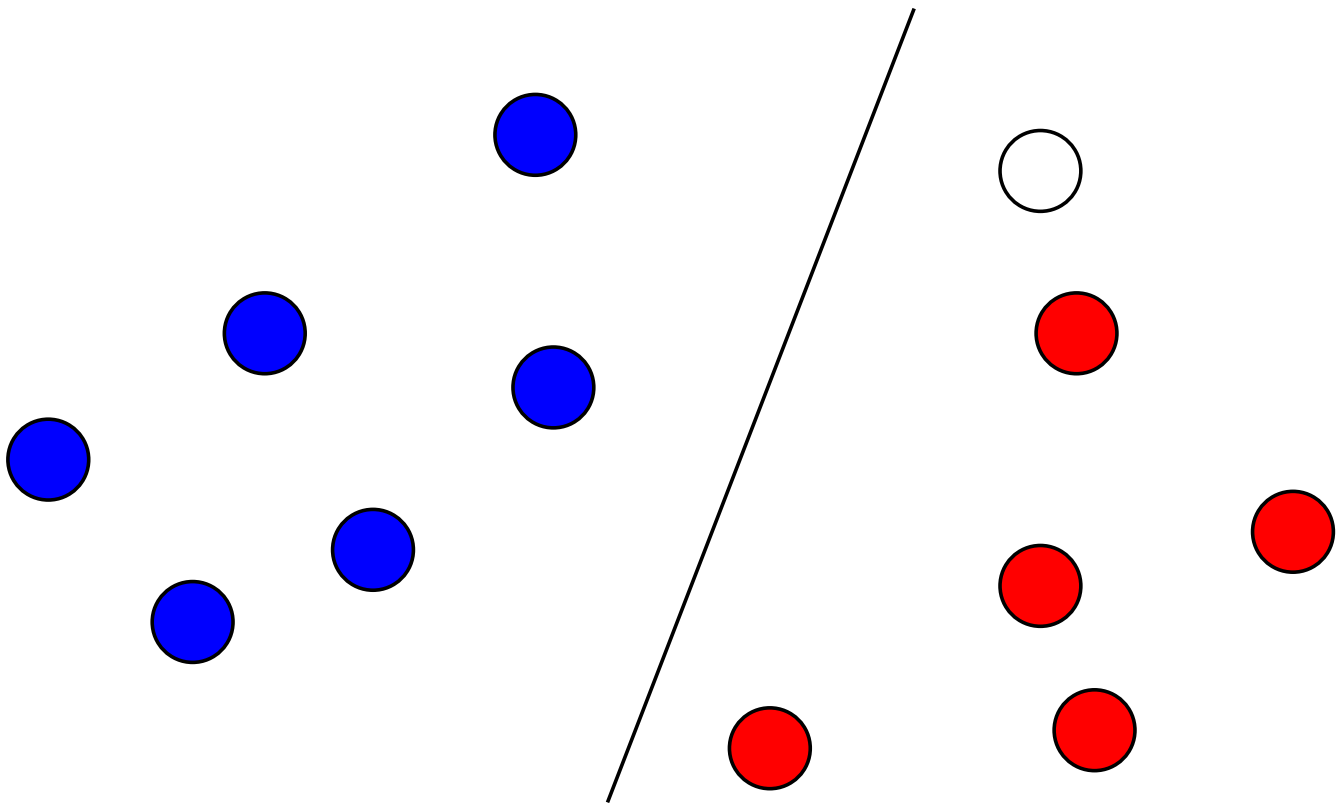
Linear classifier



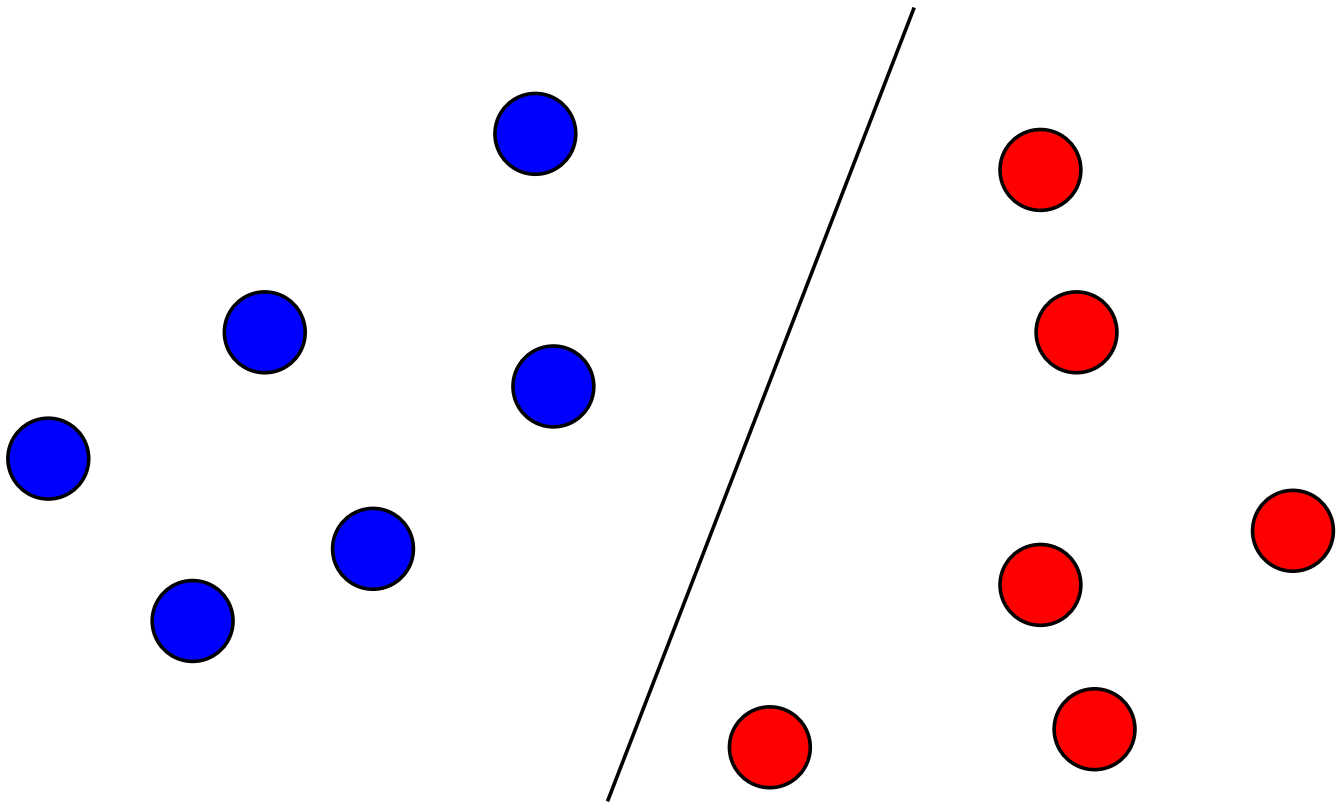
Linear classifier



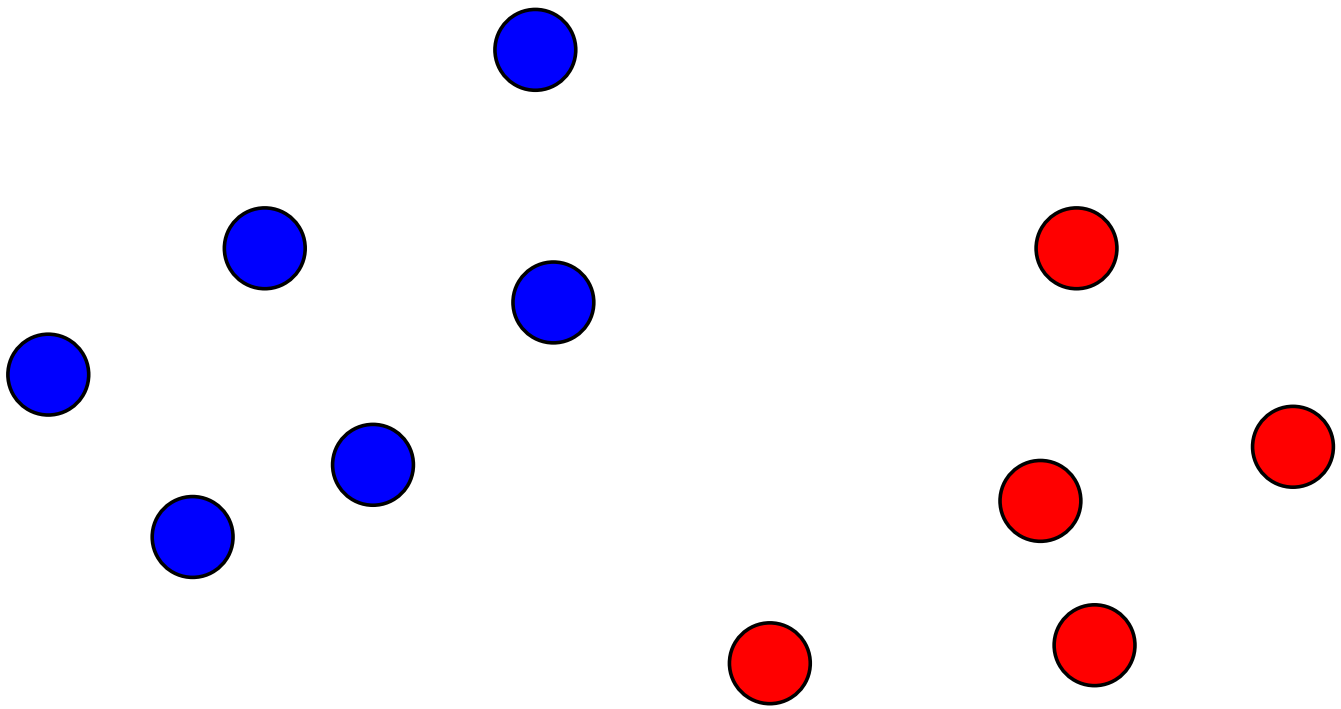
Linear classifier



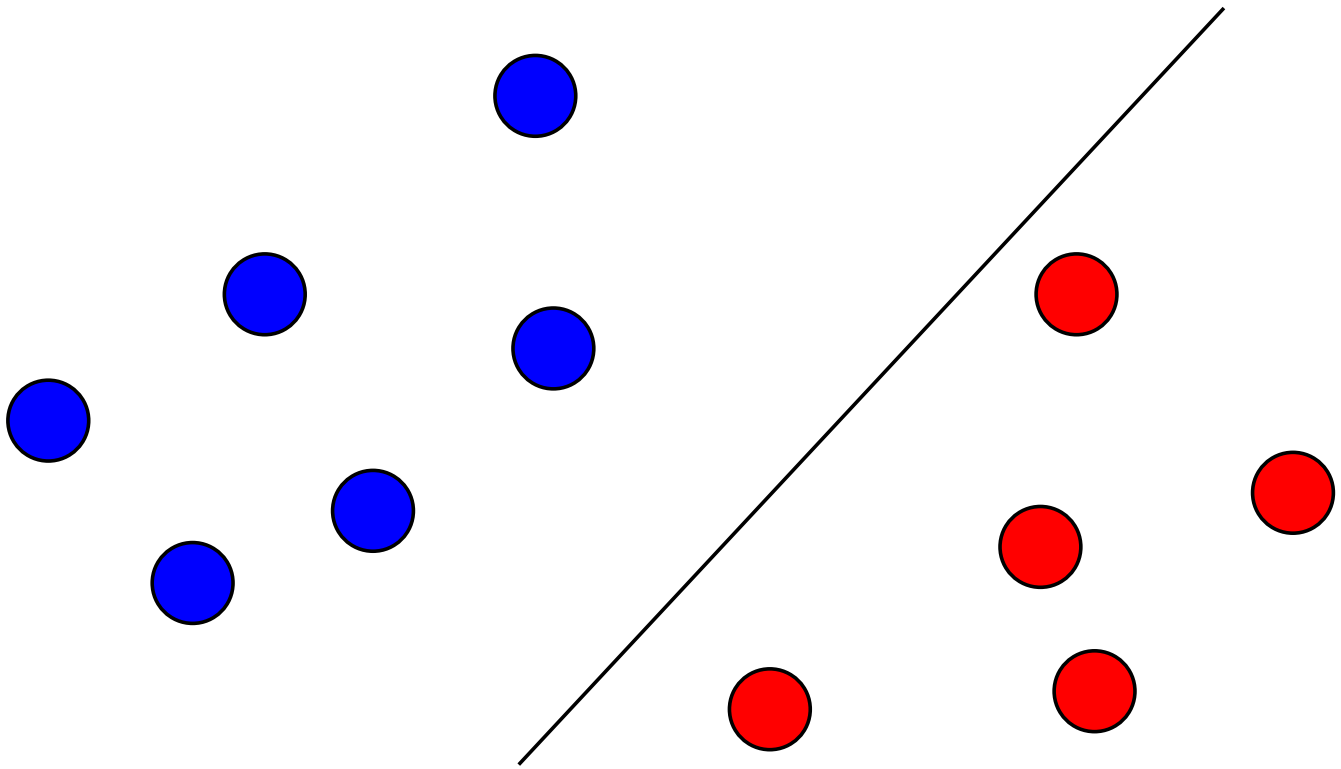
Linear classifier



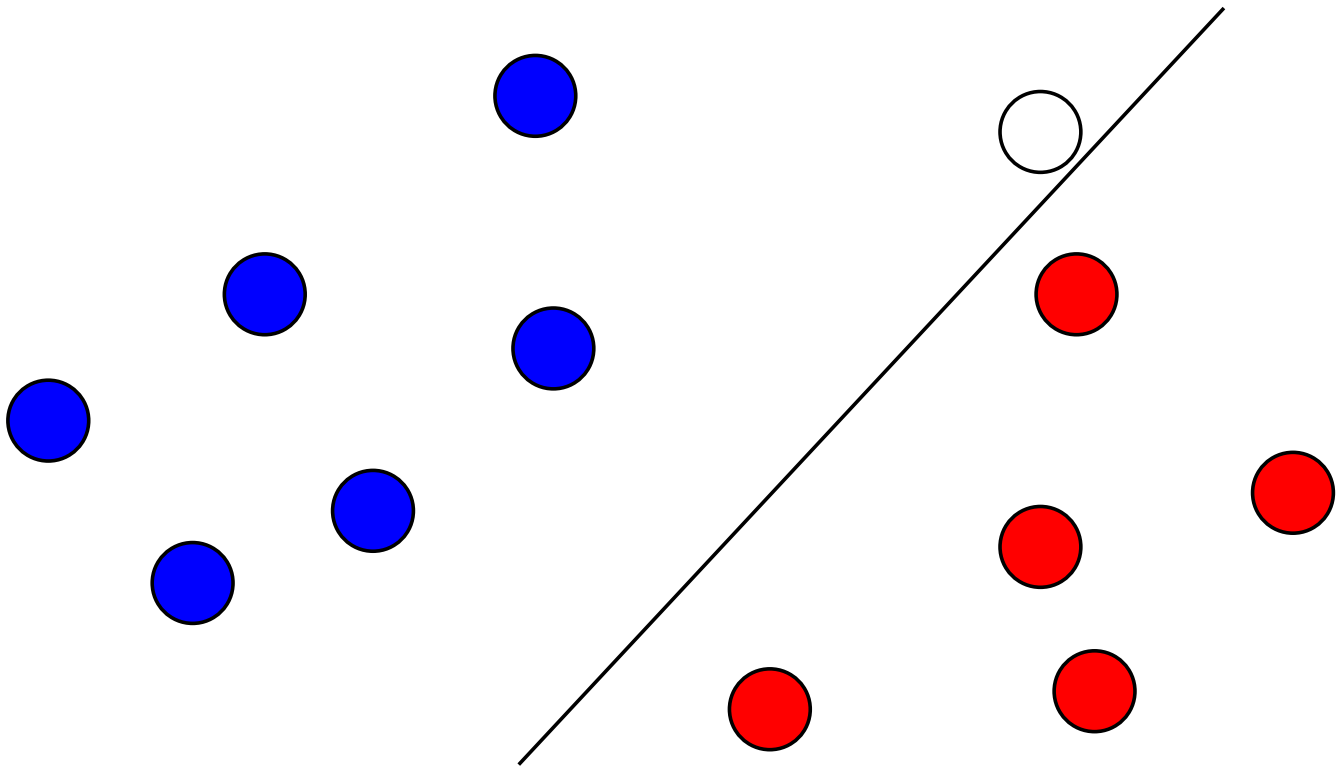
Linear classifier



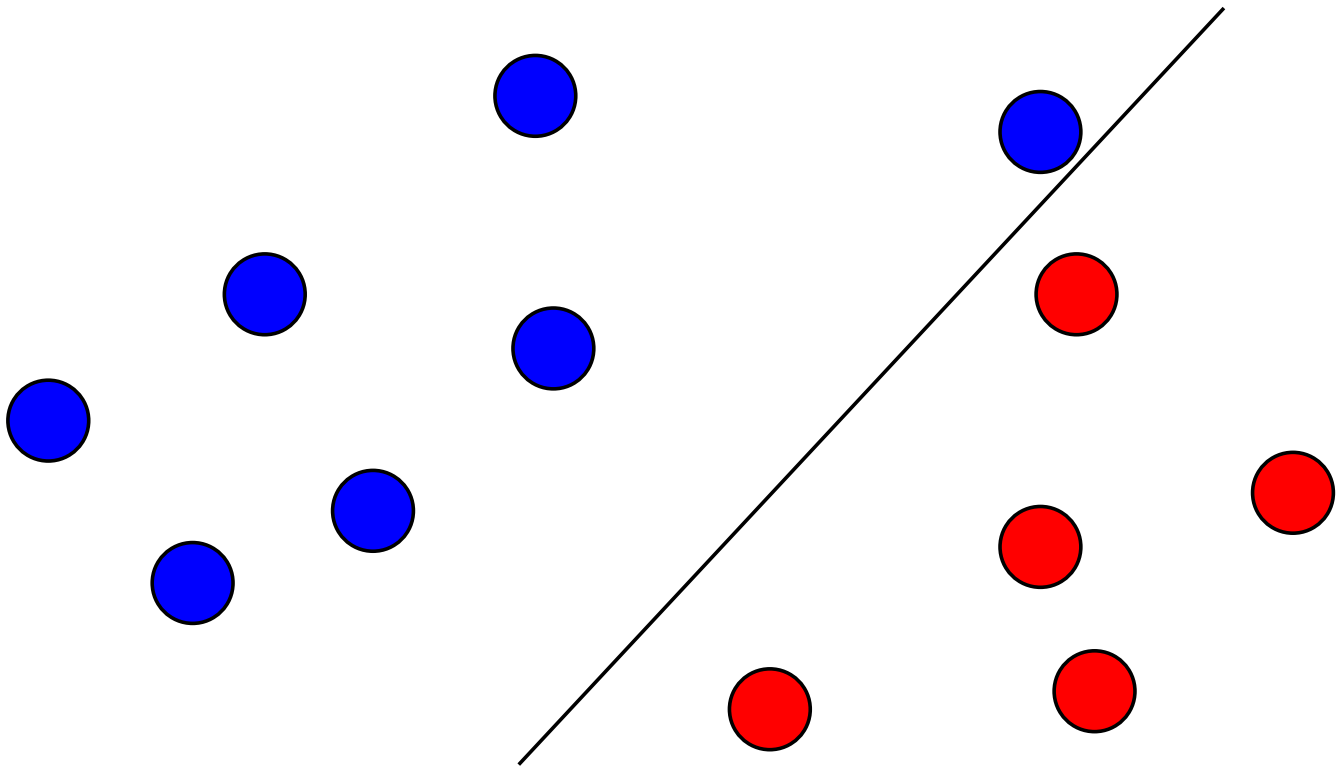
Linear classifier



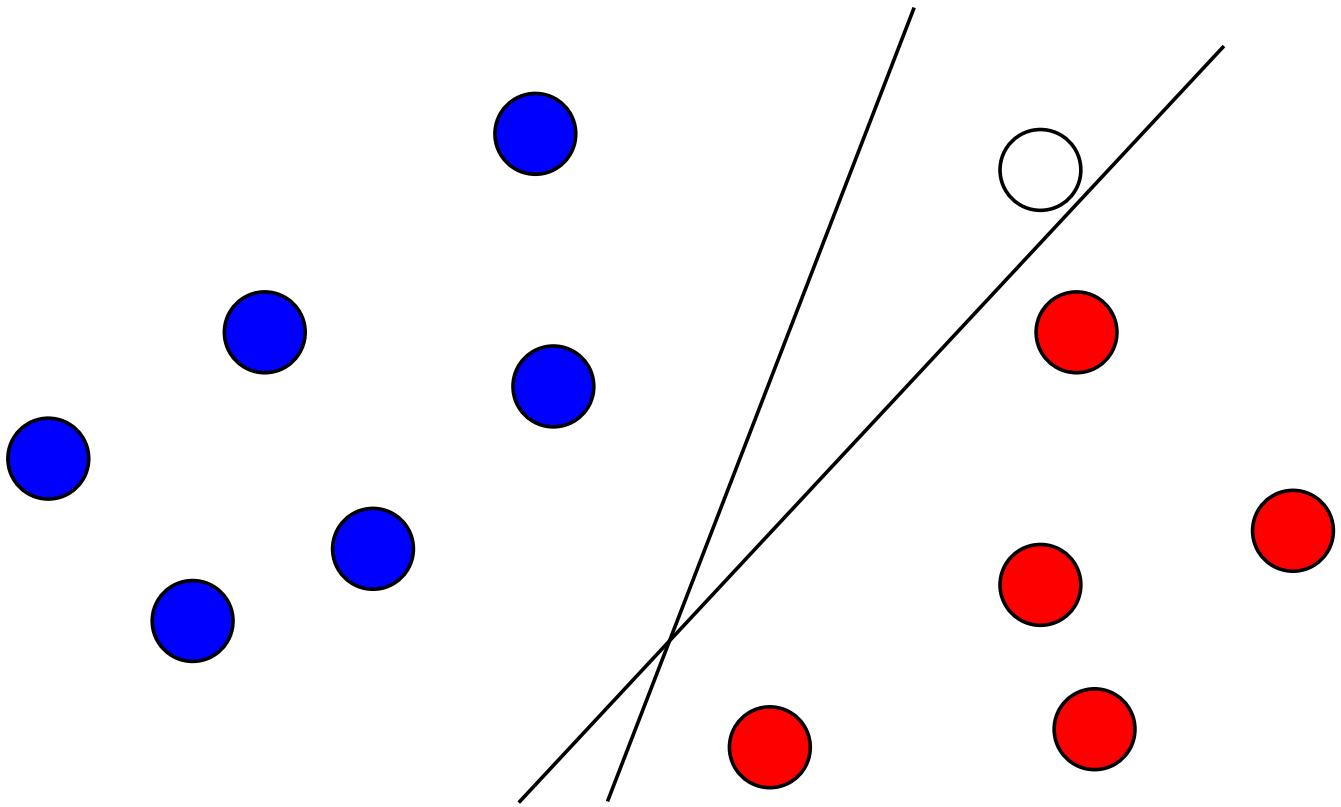
Linear classifier



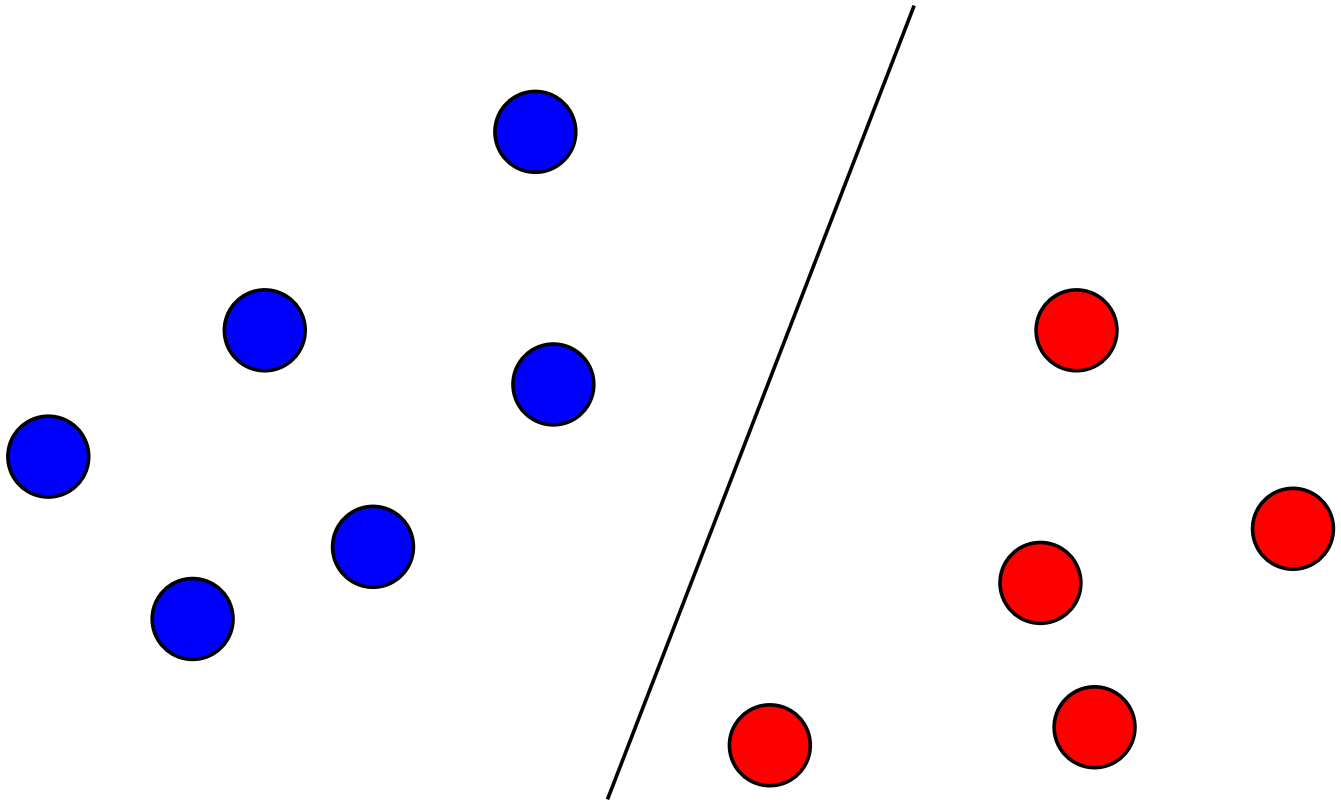
Linear classifier



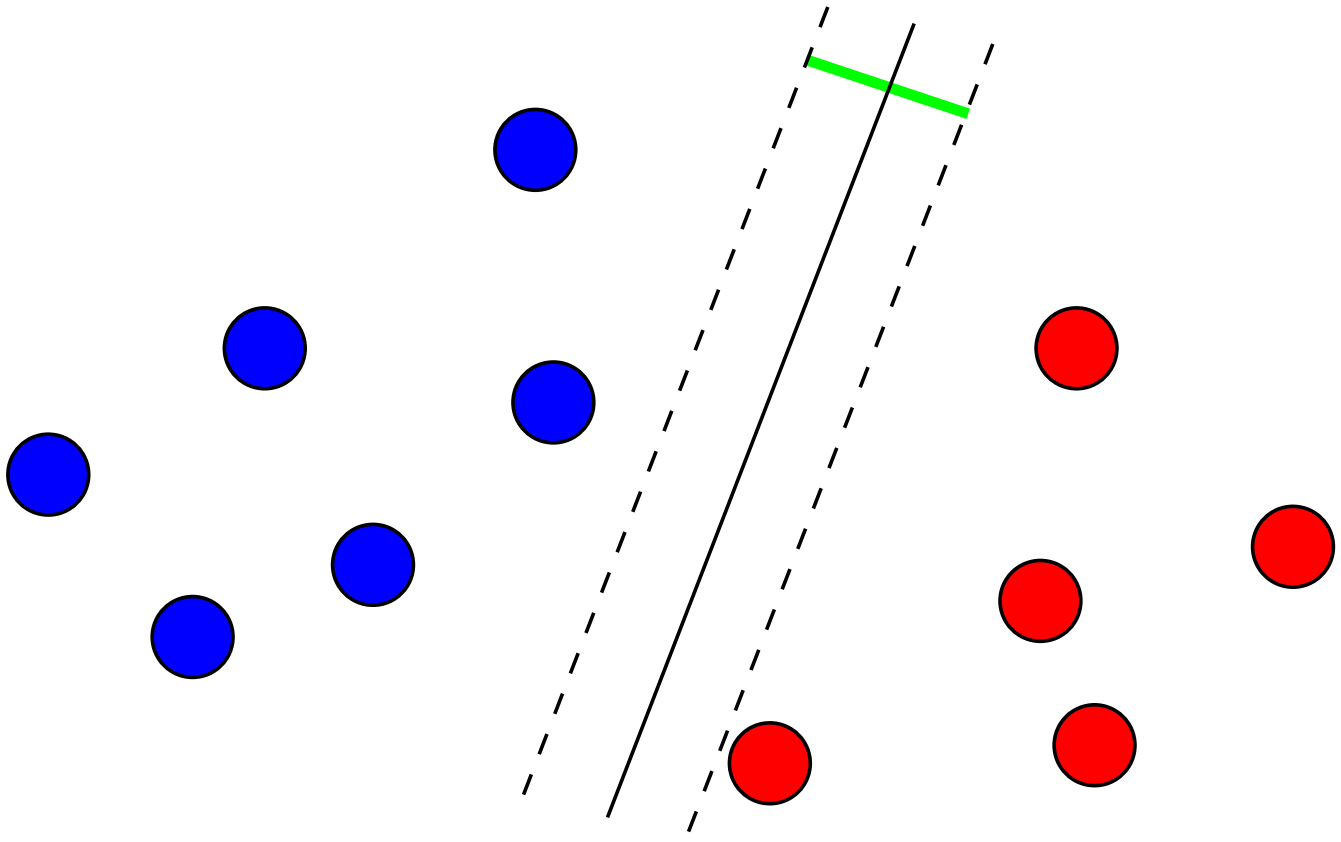
Which one is better?



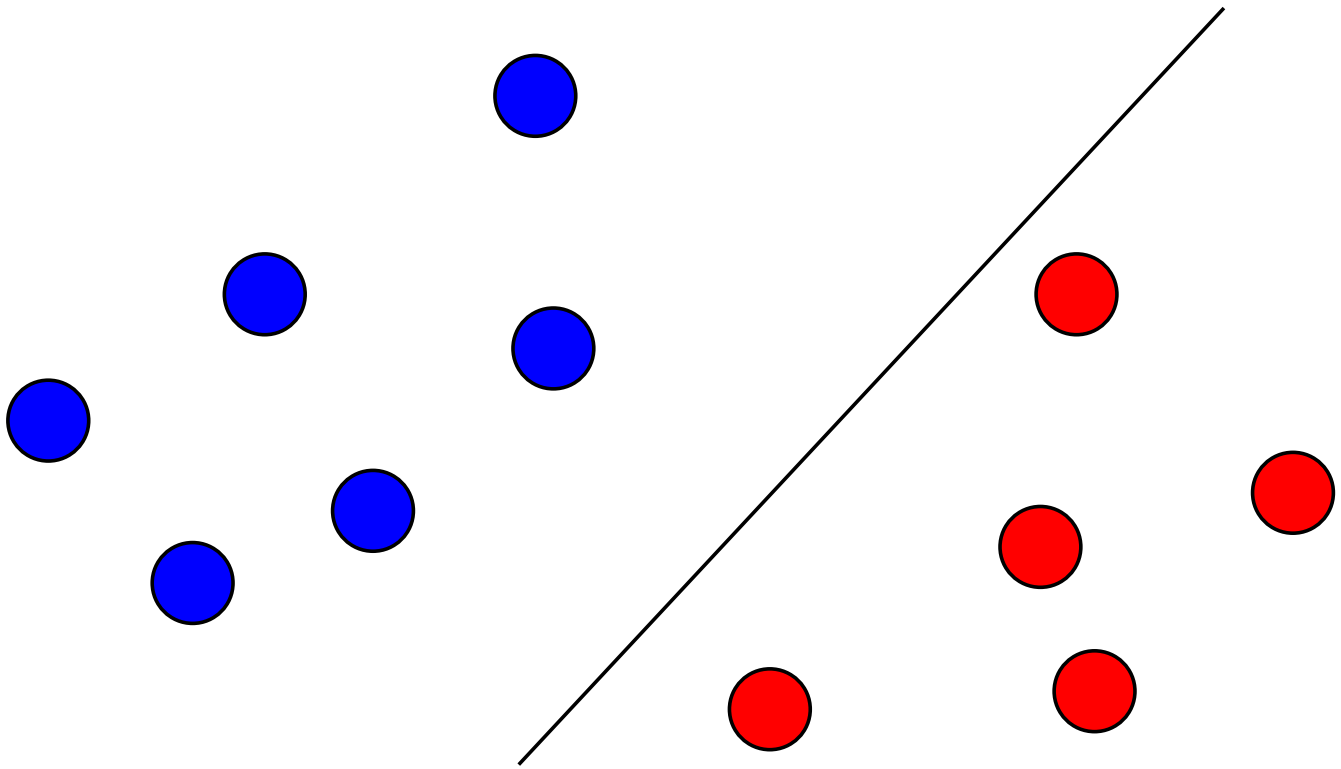
The margin of a linear classifier



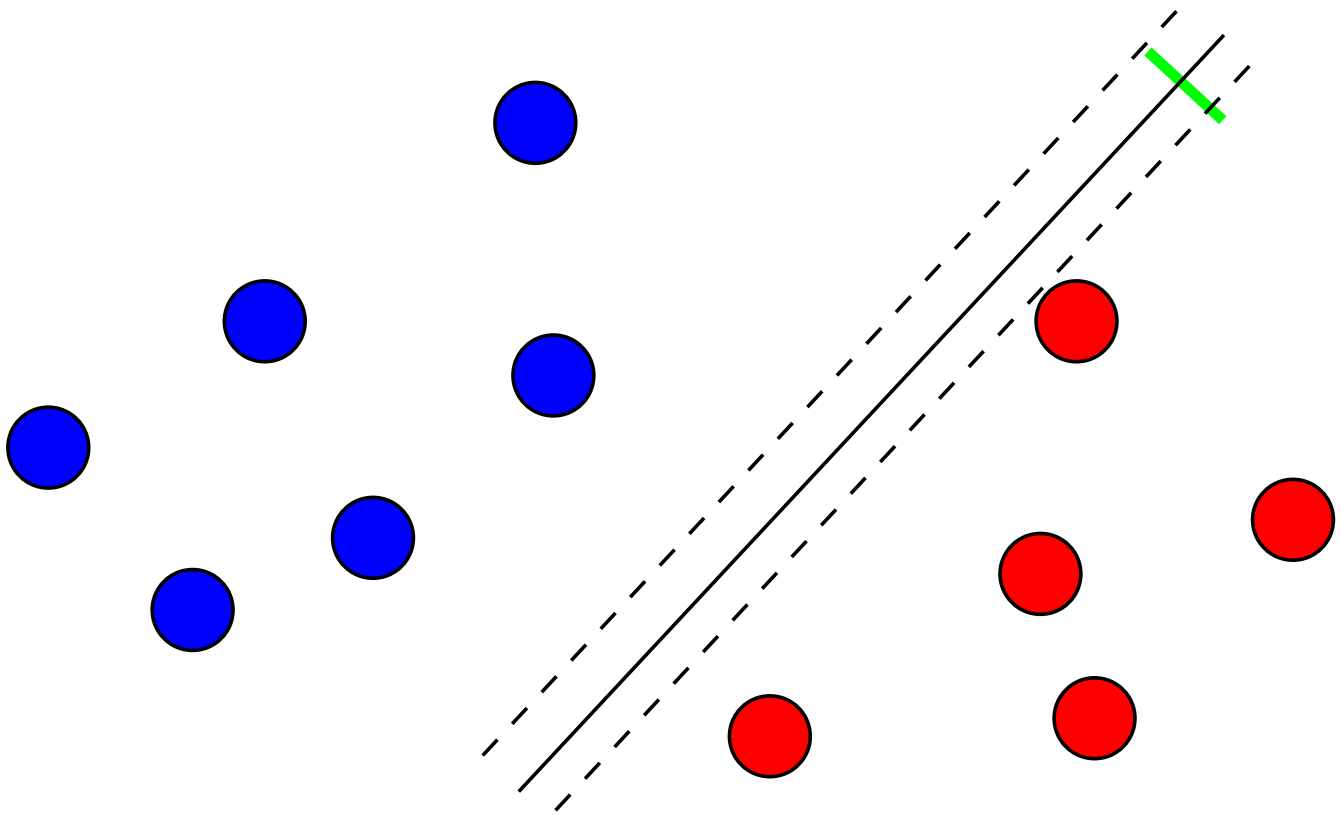
The margin of a linear classifier



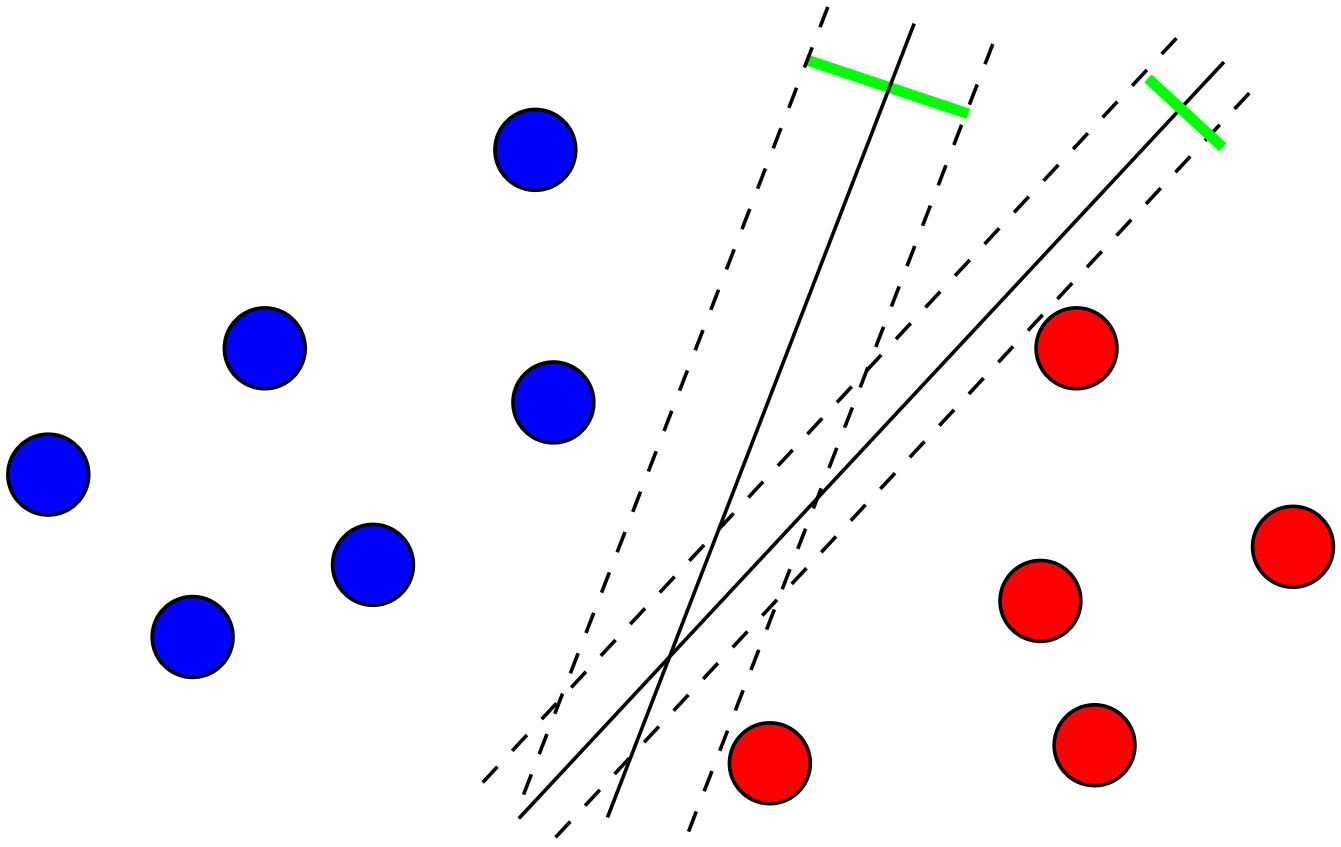
The margin of a linear classifier



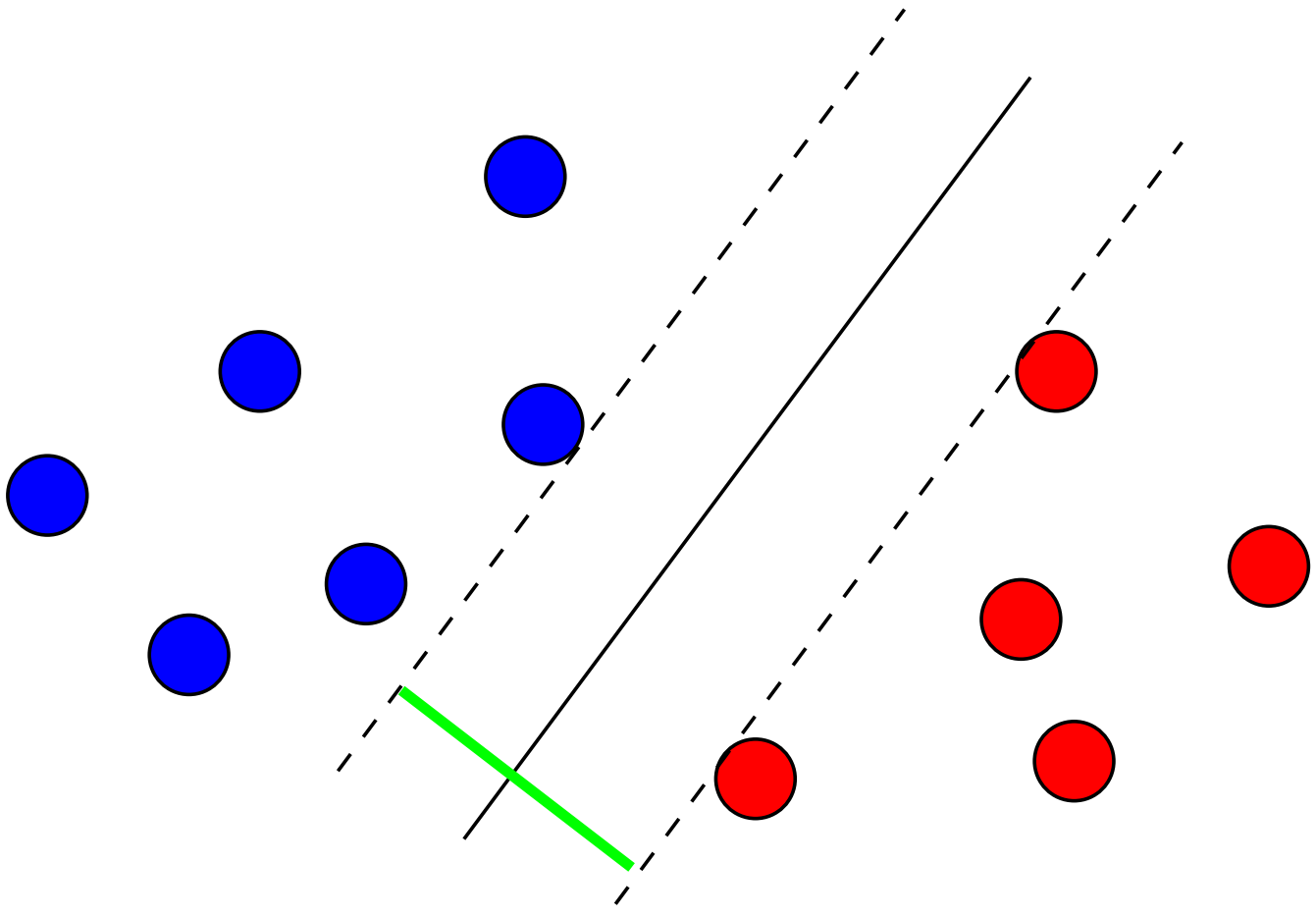
The margin of a linear classifier



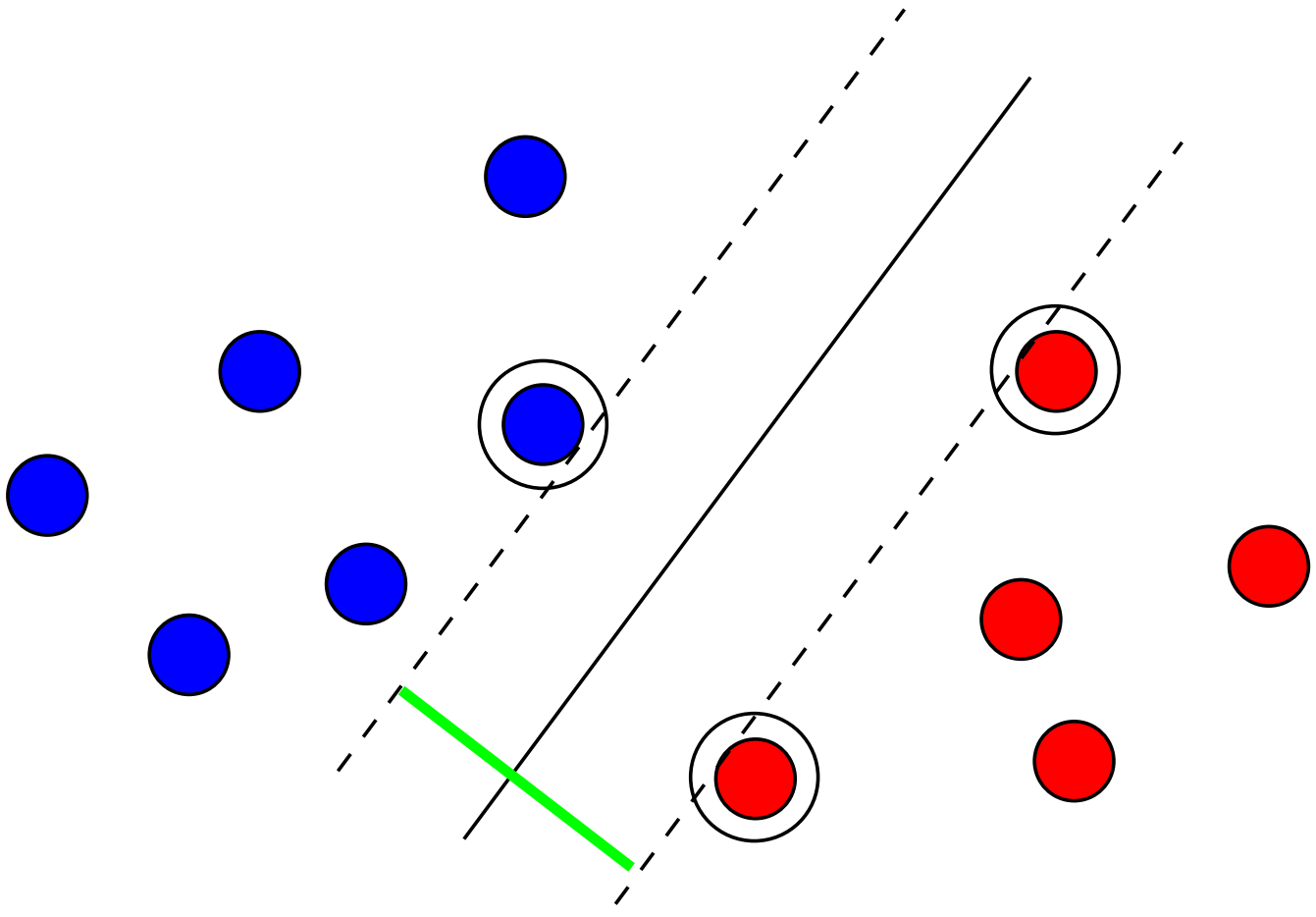
The margin of a linear classifier



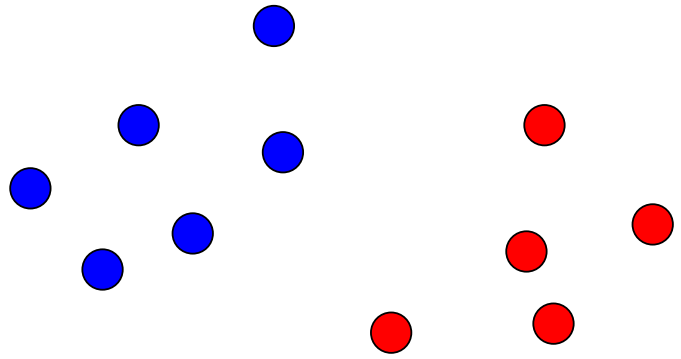
Largest margin classifier (*hard-margin SVM*)



Support vectors



More formally



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

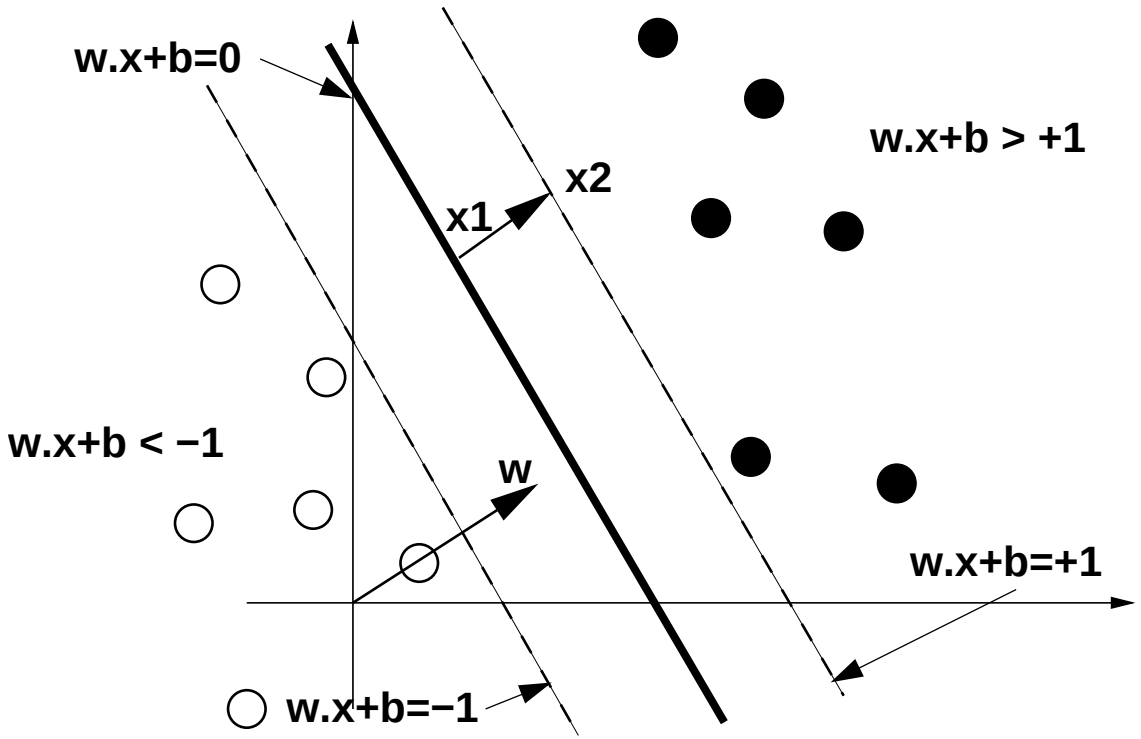
$$\mathcal{S} = \left\{(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)\right\} ,$$

  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:

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

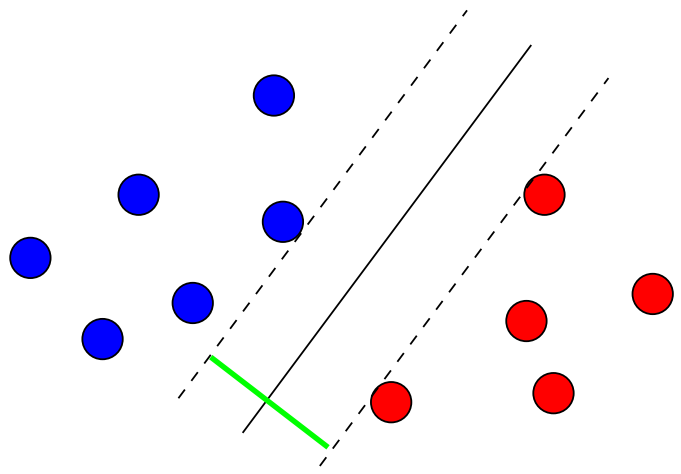
For negative examples ($y_i = -1$) this means:

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

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 α_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

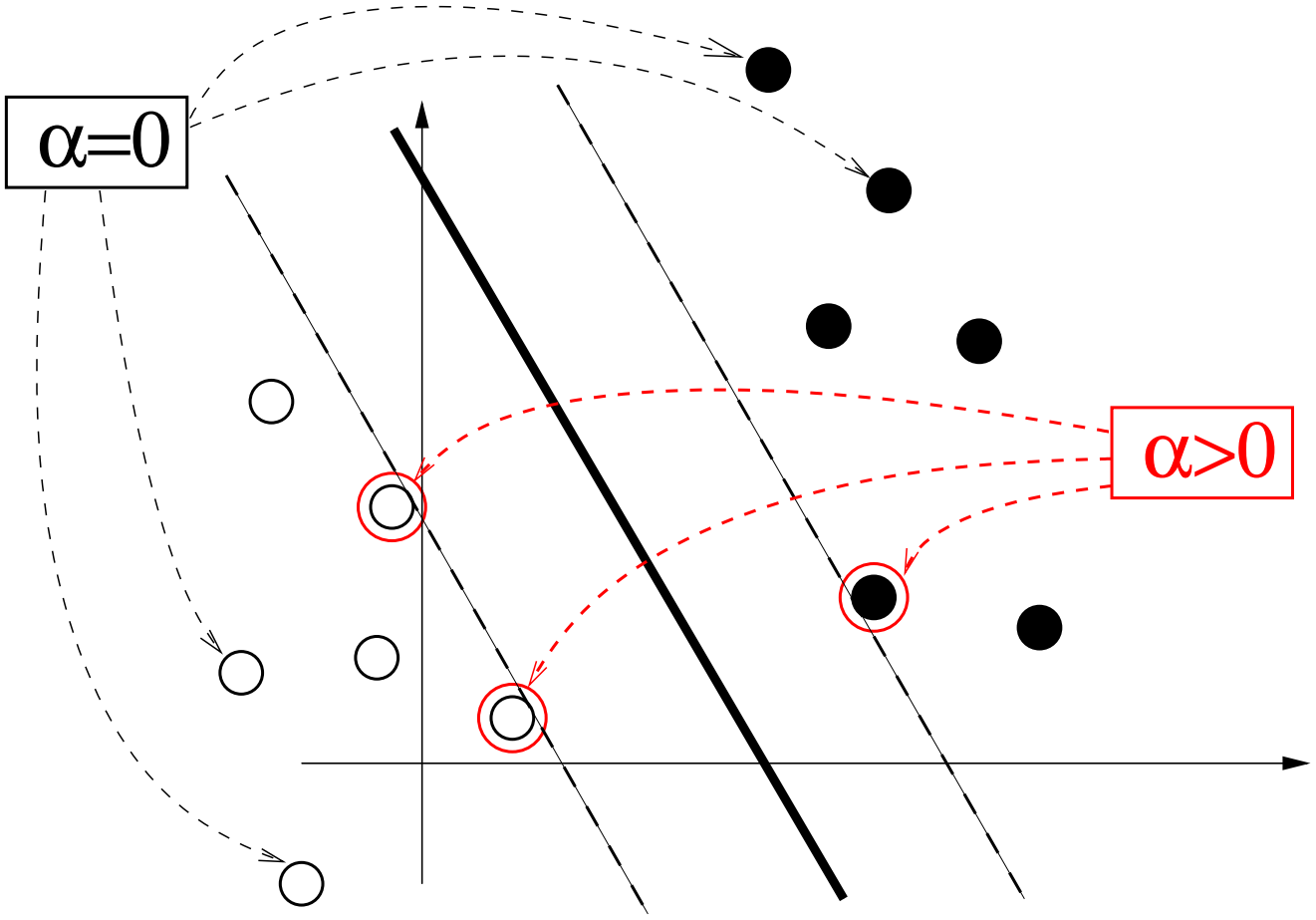
Once $\vec{\alpha}^*$ is found, we recover $(\vec{w}^*, b^*)$ corresponding to the optimal hyperplane. w^* is given by:

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

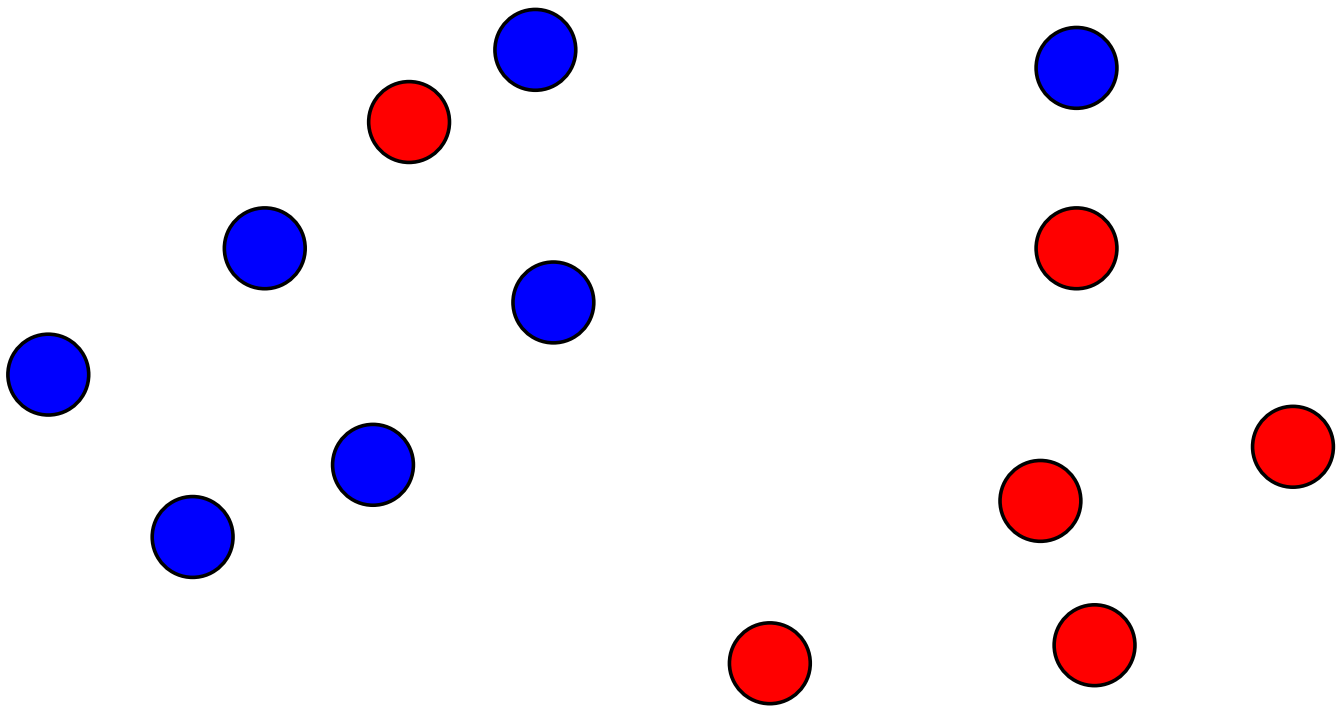
and the decision function is therefore:

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

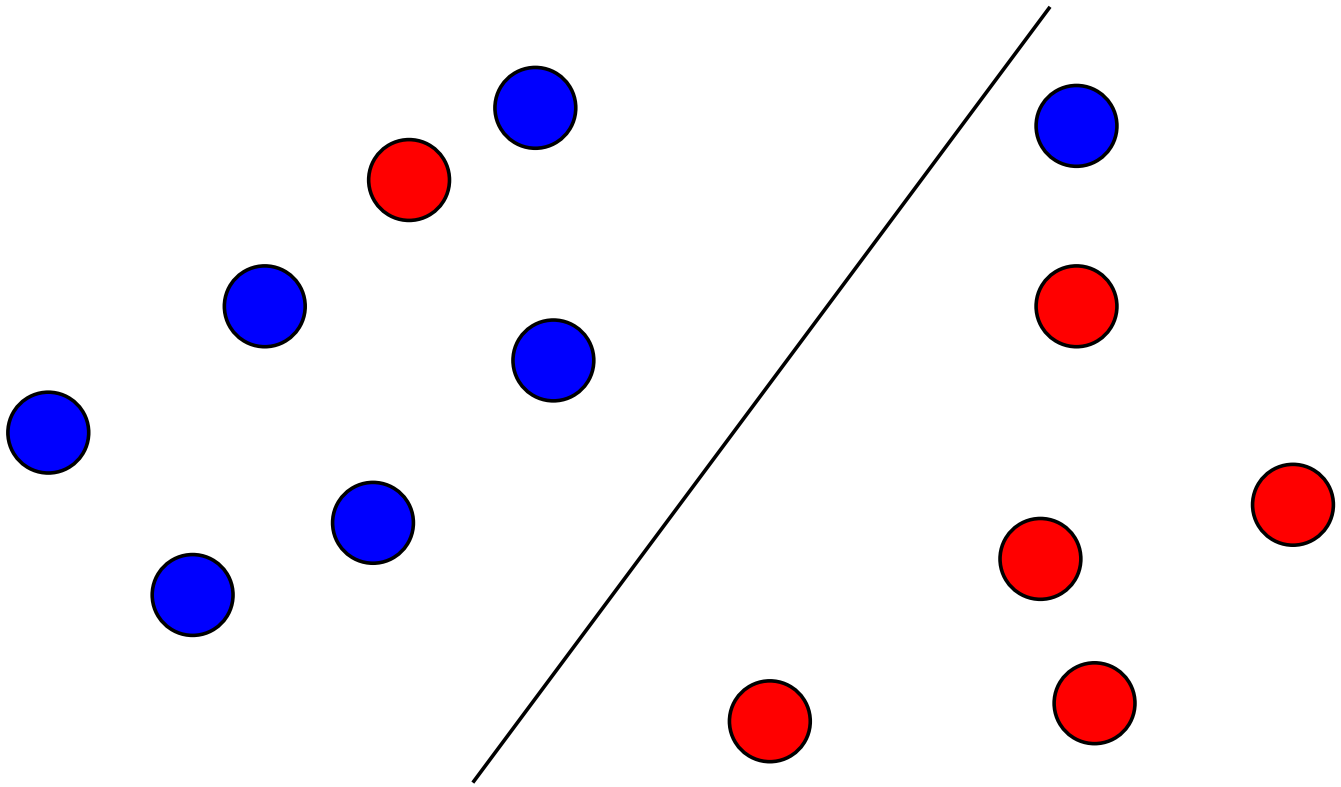
Interpretation: support vectors



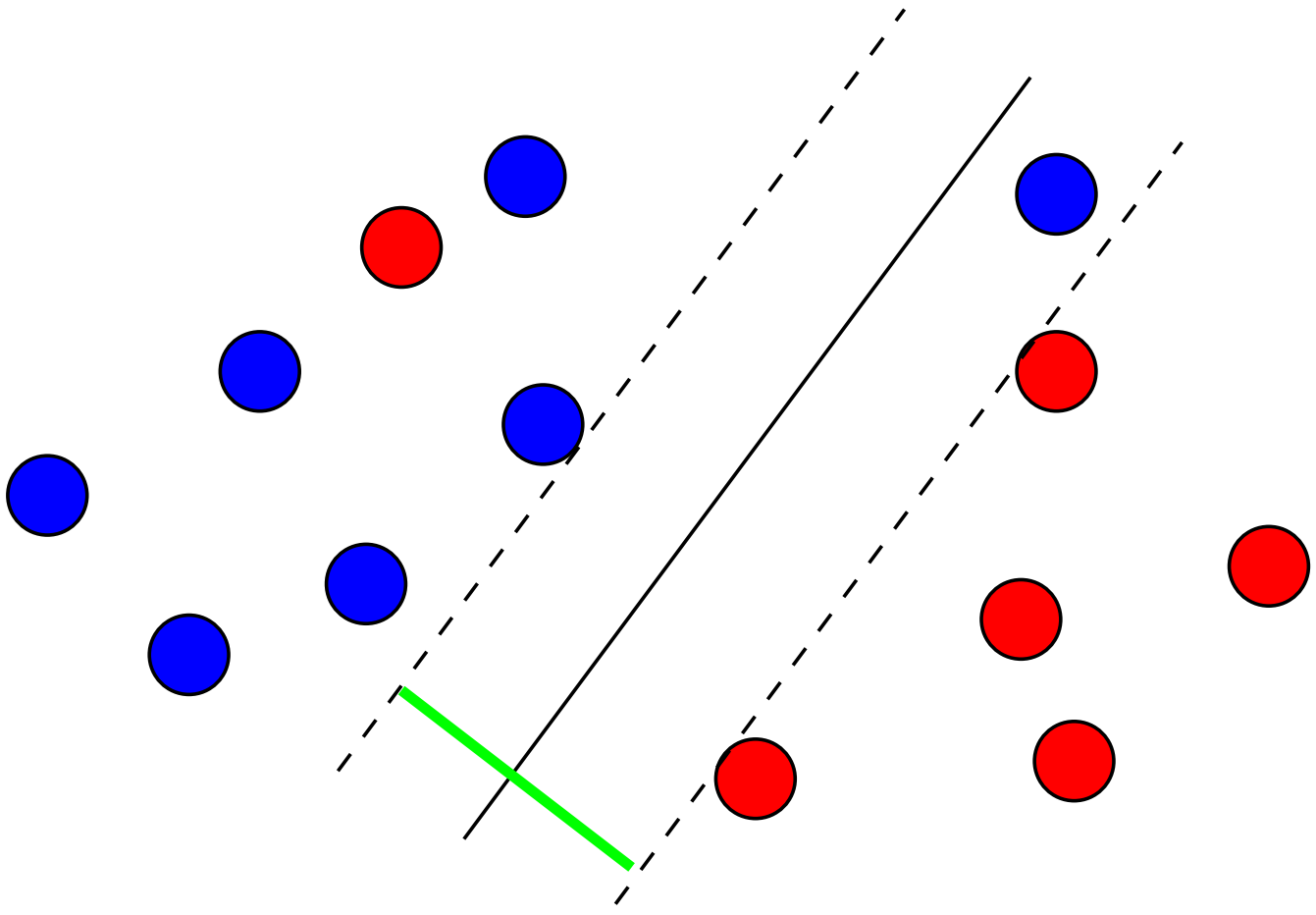
What if data are not linearly separable?



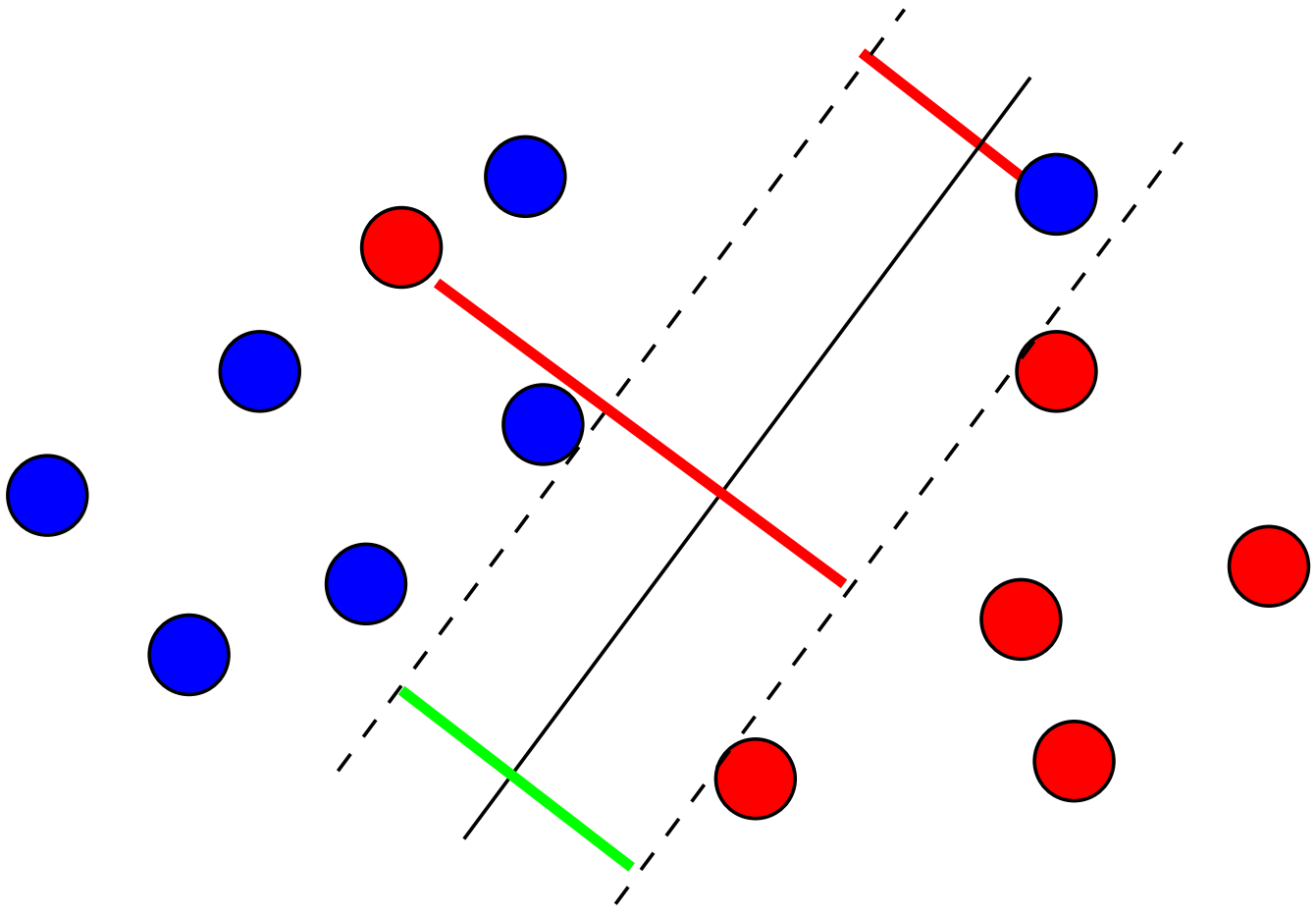
What if data are not linearly separable?



What if data are not linearly separable?



What if data are not linearly separable?

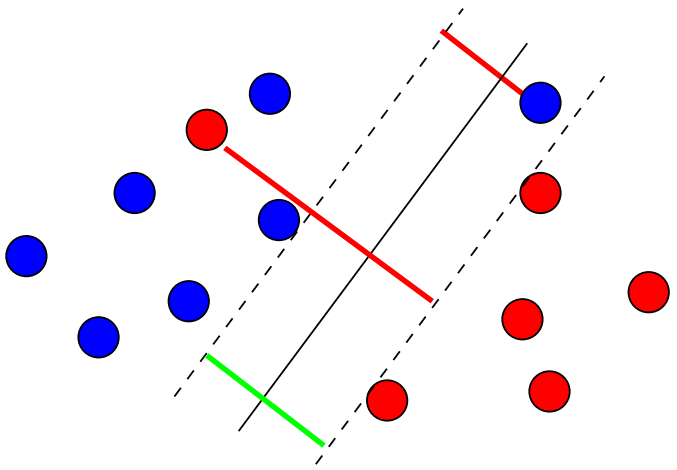


Soft-margin SVM

- 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



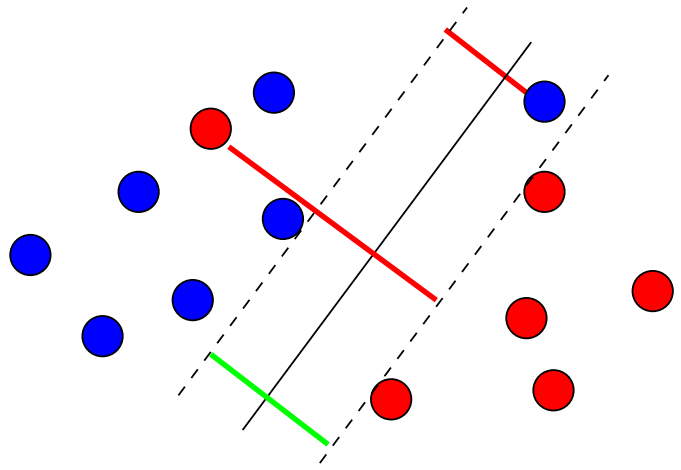
Soft-margin SVM formulation

- 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:

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

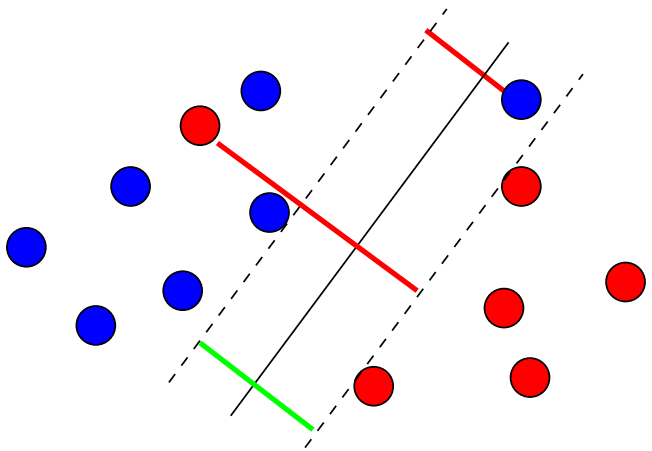


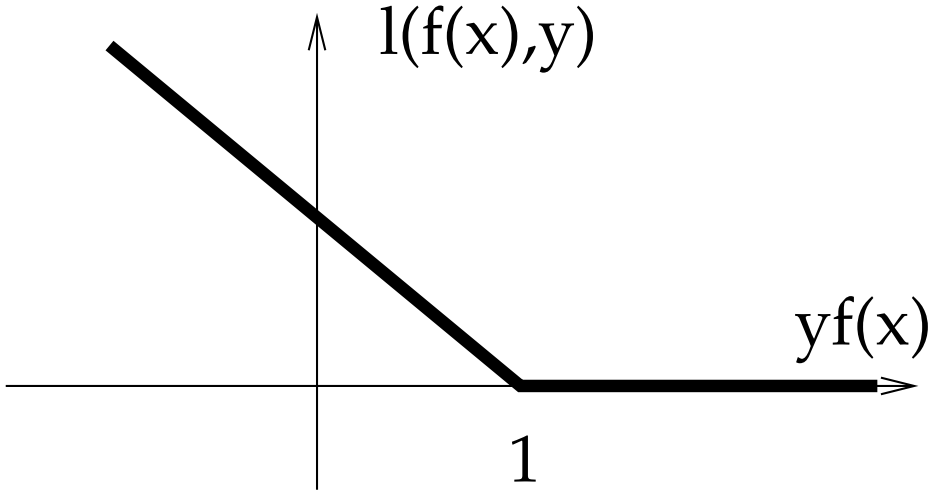
Soft-margin SVM and hinge loss

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

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*)

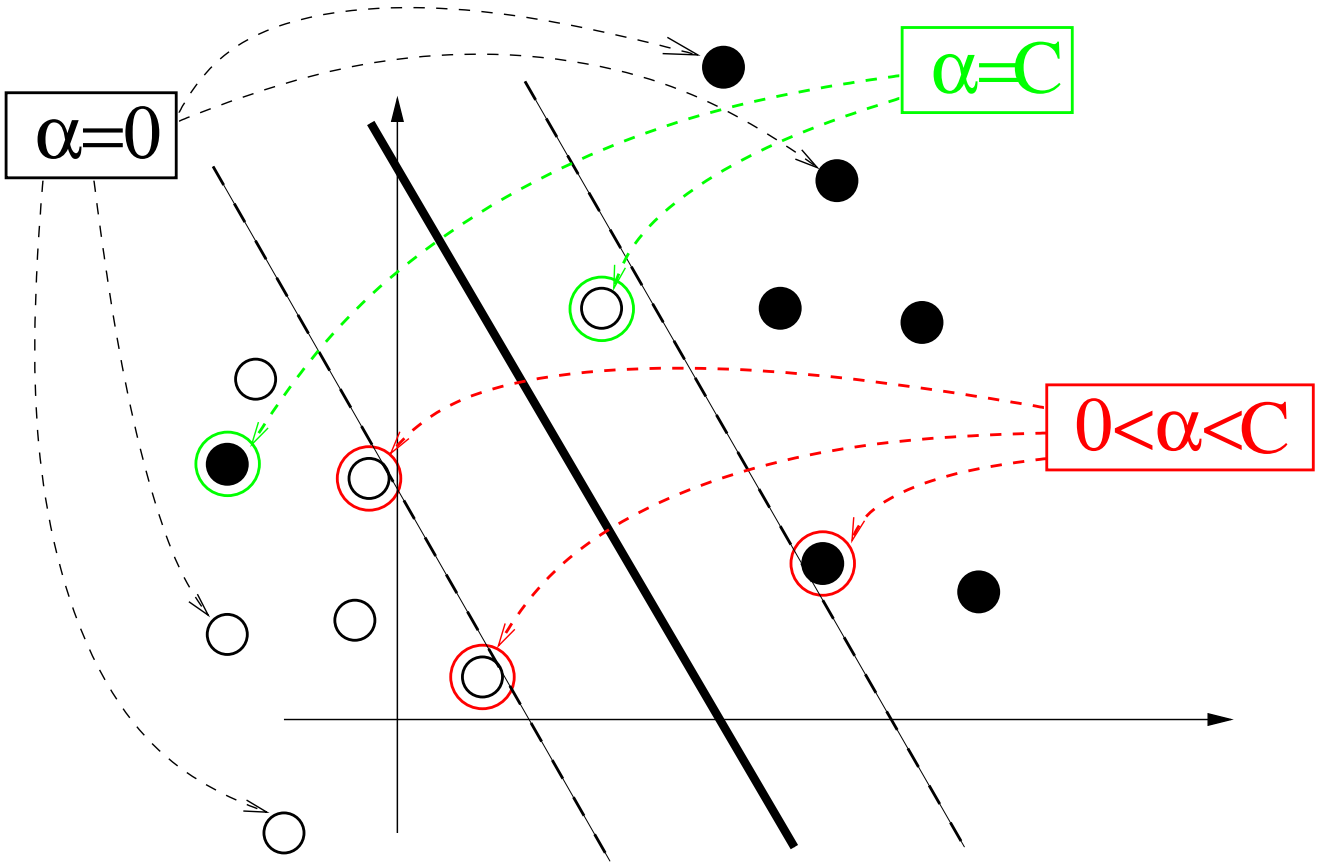
Maximize

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

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

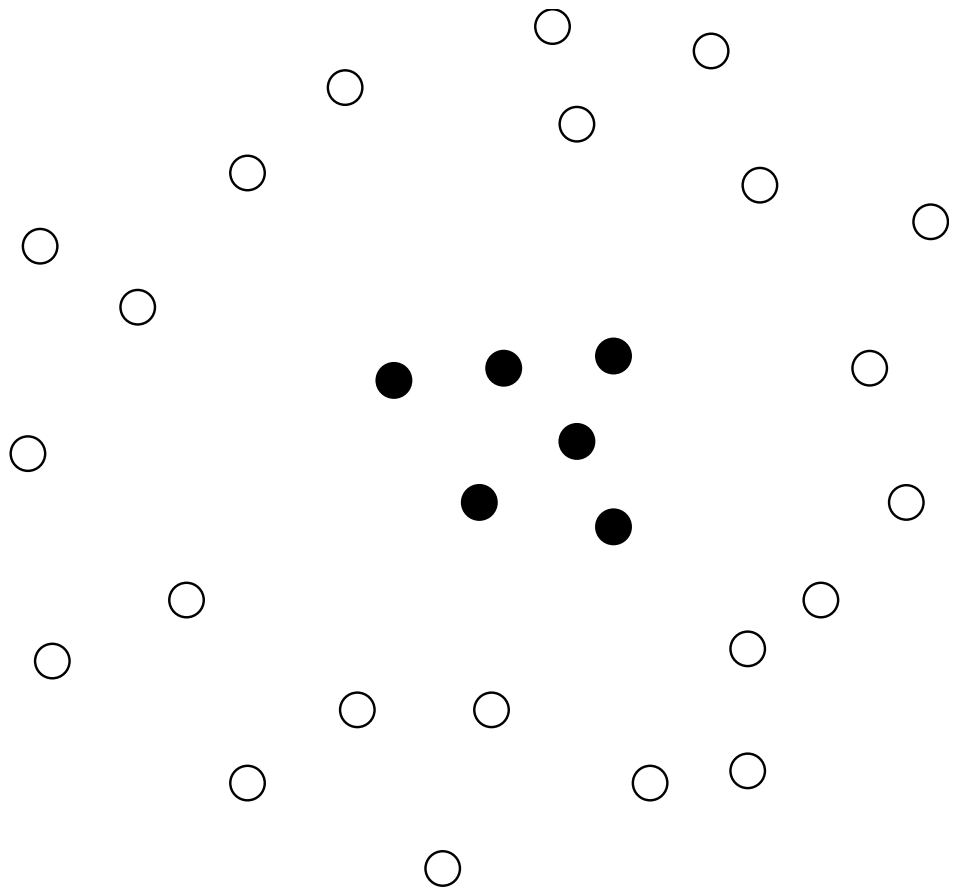
$$L(\vec{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \vec{x}_i . \vec{x}_j \, ,$$

under the constraints:

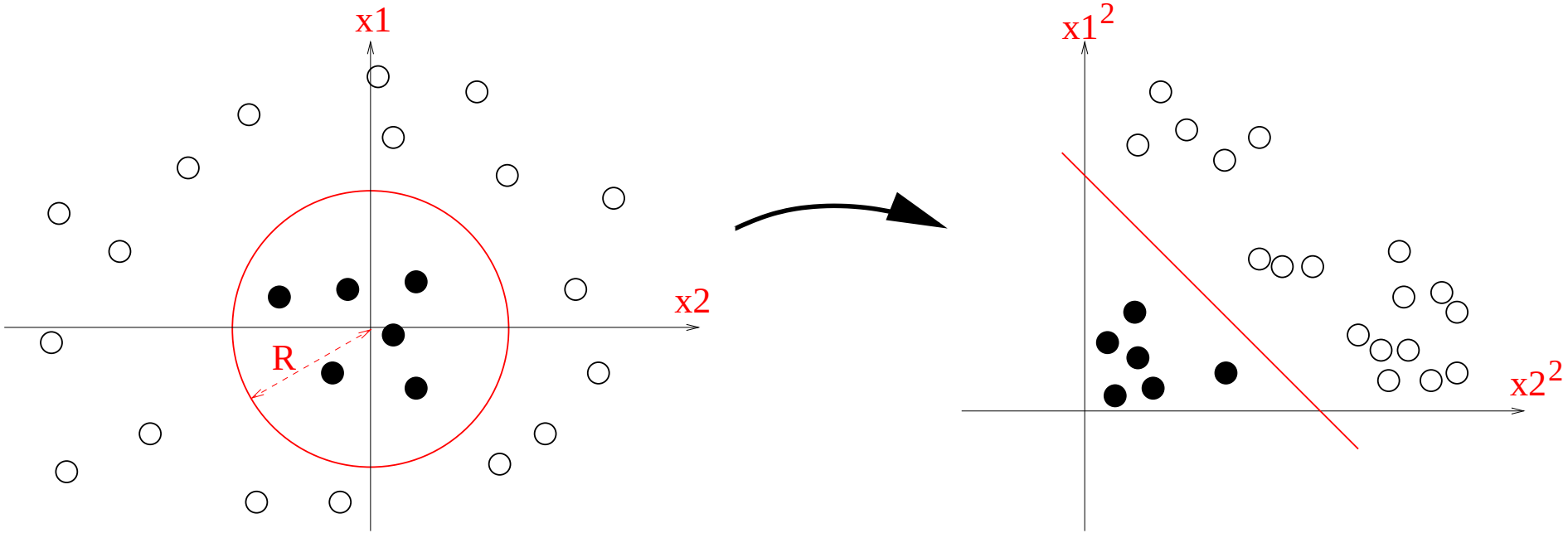
$$\begin{cases} 0 \leq \alpha_i \leq C, \quad \text{for } i = 1, \ldots, n \\ \sum_{i=1}^{n} \alpha_i y_i = 0. \end{cases}$$

Outline

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

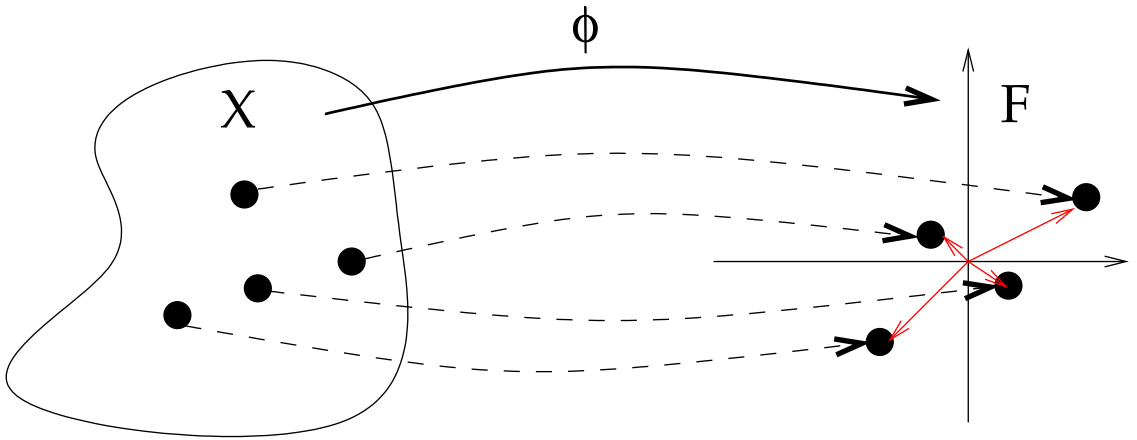
Definition

For a given mapping

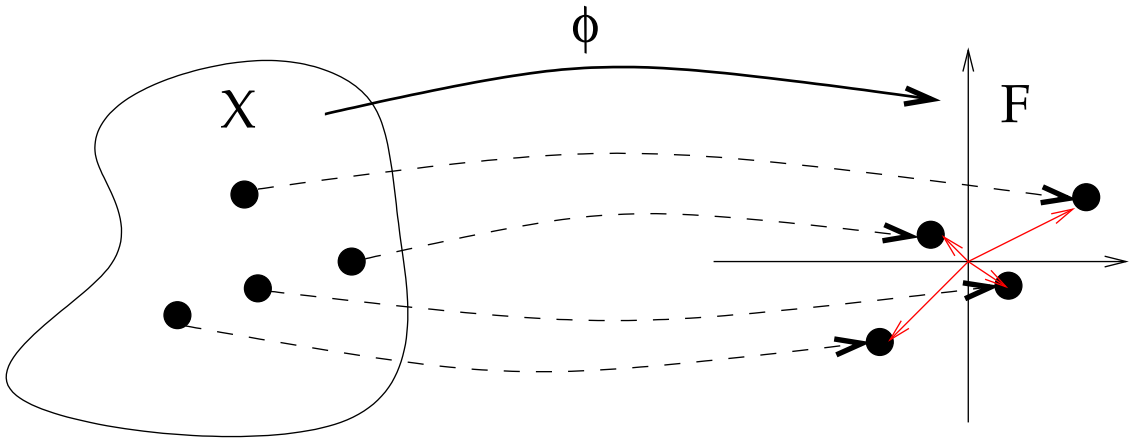
$$\Phi : \mathcal{X} \mapsto \mathcal{H}$$

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

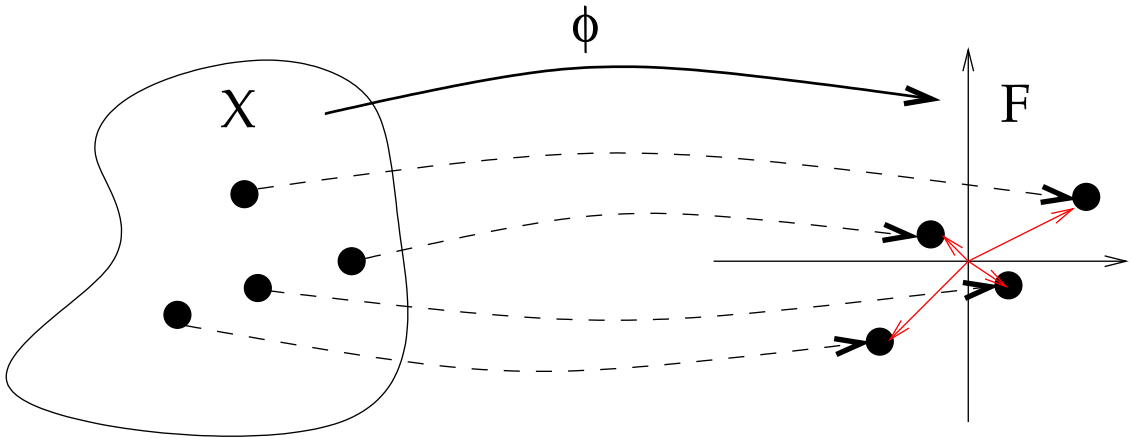


Let $\mathcal{X} = \mathcal{H} = \mathbb{R}^2$ and for $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ let $\Phi(x) = \begin{pmatrix} x_1^2 \\ x_2^2 \end{pmatrix}$

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

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

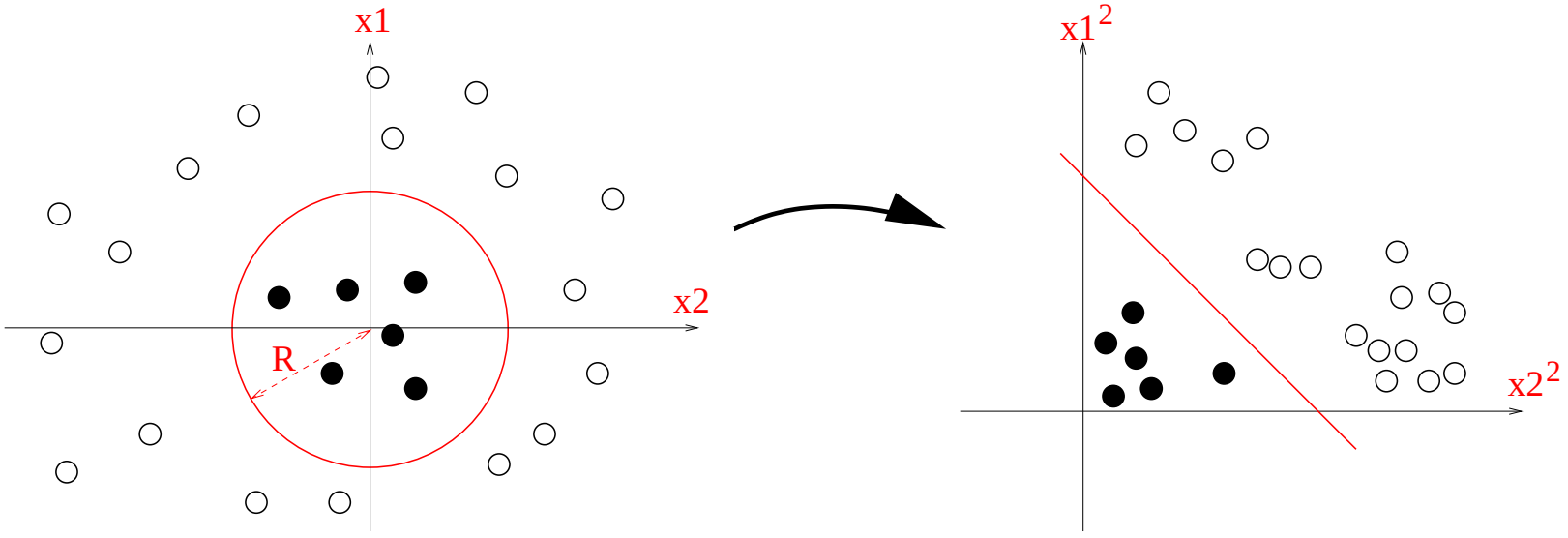
  under the constraints:

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

- Predict with the decision function

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

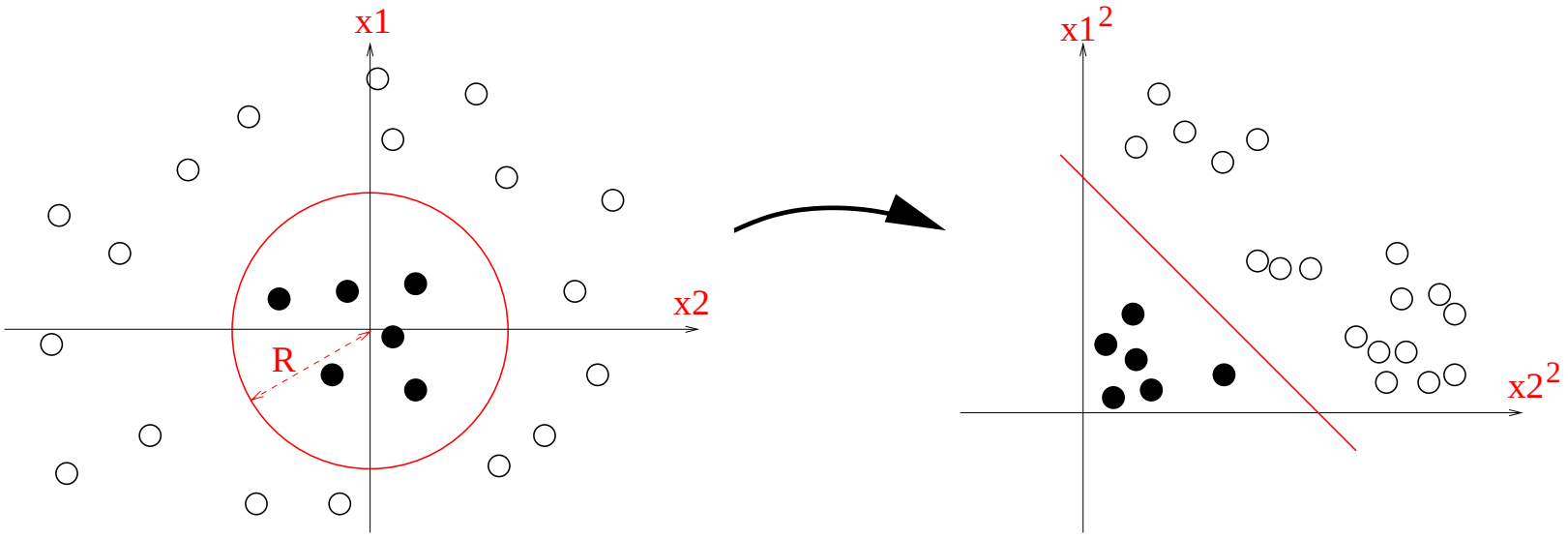
Trick 2 illustration: polynomial kernel



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)
```

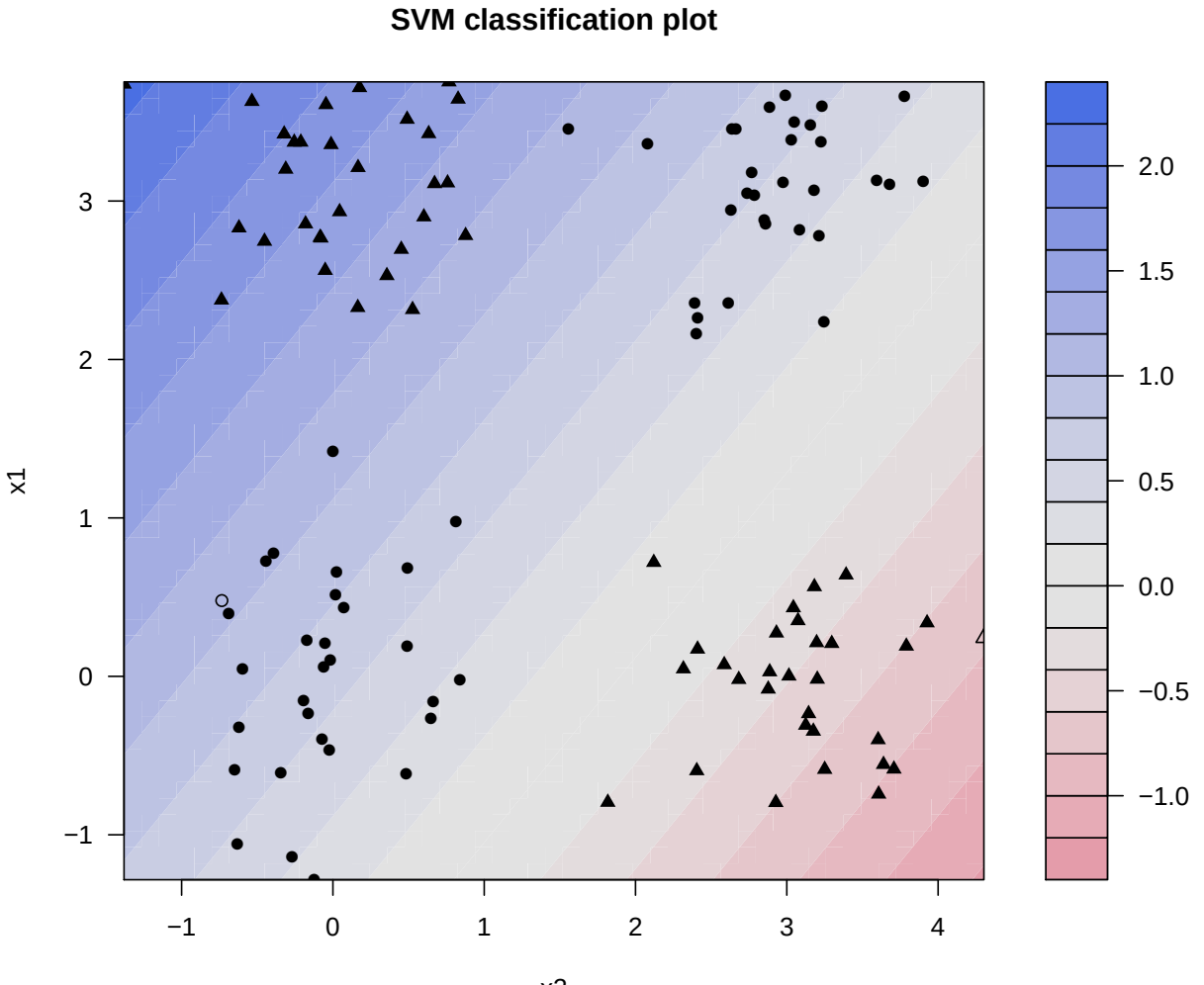
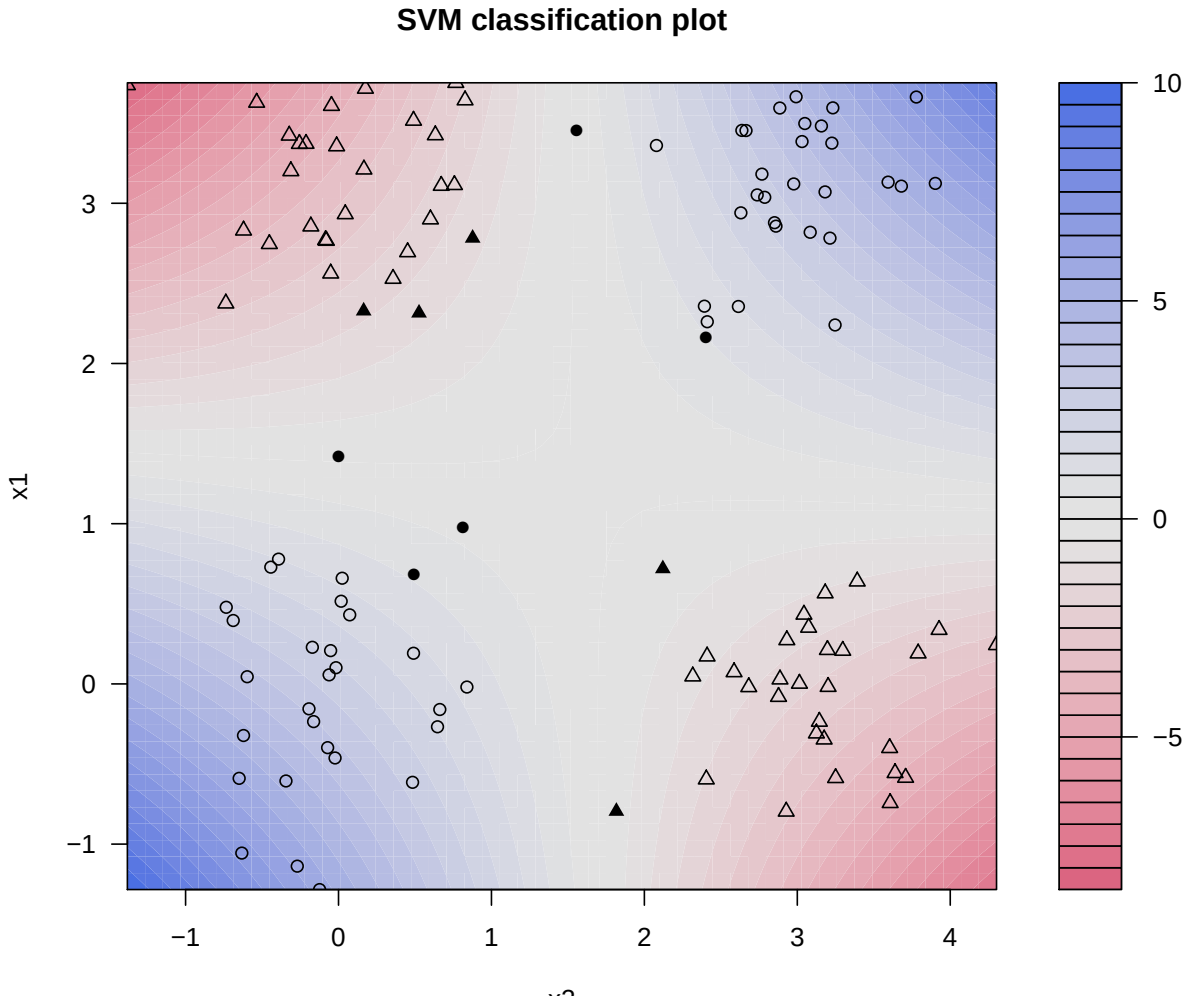


Illustration: toy nonlinear problem, polynomial SVM

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



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

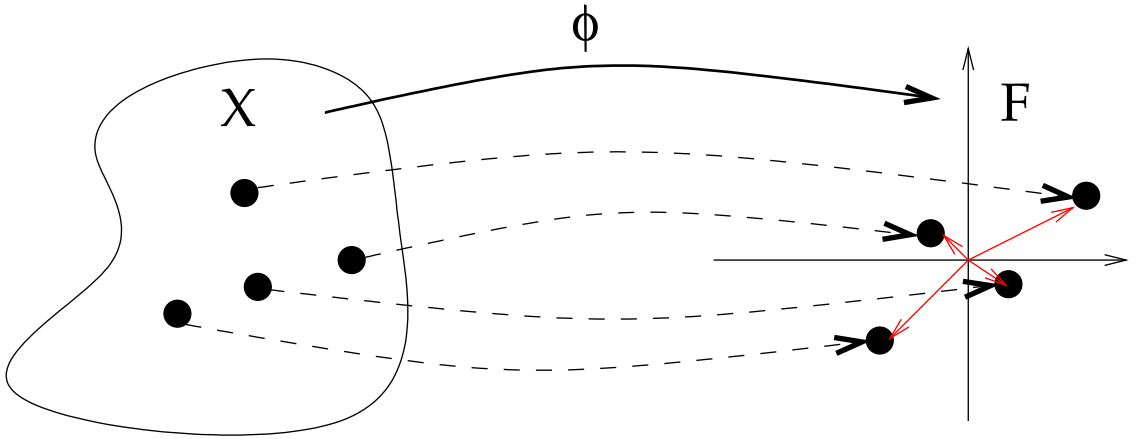
Definition

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:

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

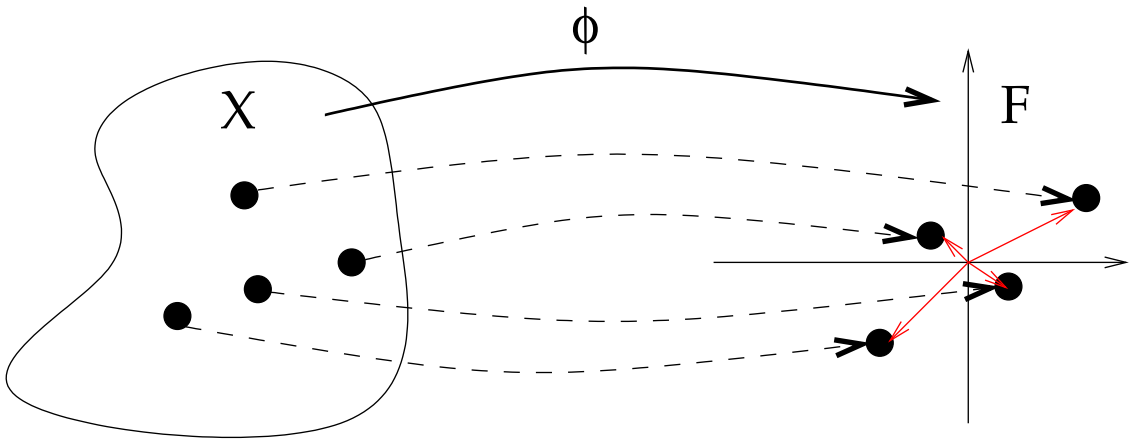
and which satisfies, for all $N \in \mathbb{N}$, $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) \in \mathcal{X}^N$ et $(a_1, a_2, \ldots, a_N) \in \mathbb{R}^N$:

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

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 (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

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:

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

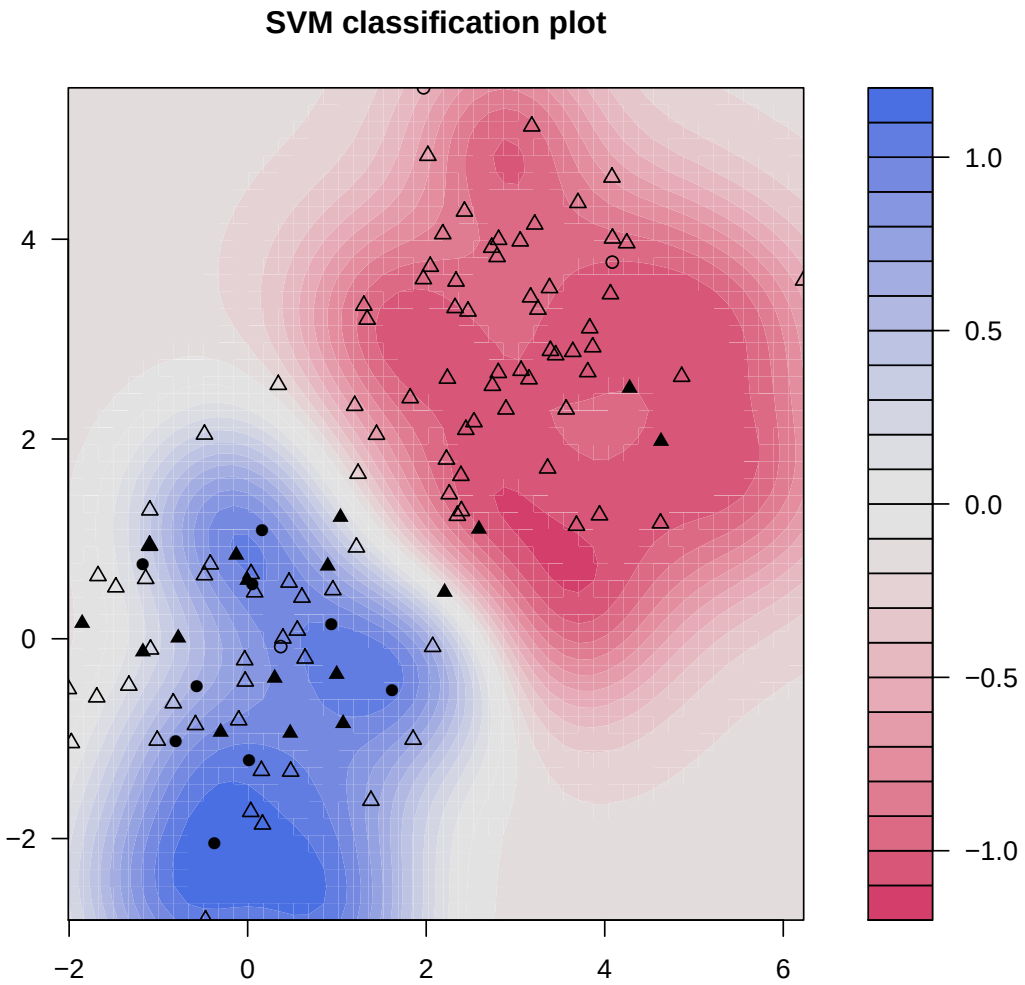
$$\text{s.t. } 0 \leq \alpha_i \leq C, \quad \text{and } \sum_{i=1}^{n} \alpha_i y_i = 0.$$

- Prediction

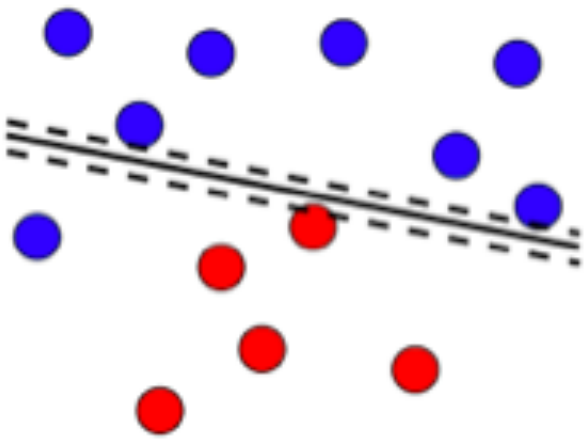
$$f(\vec{x}) = \sum_{i=1}^{n} \alpha_i \exp\left(-\frac{||\vec{x} - \vec{x}_i||^2}{2\sigma^2} \right)$$

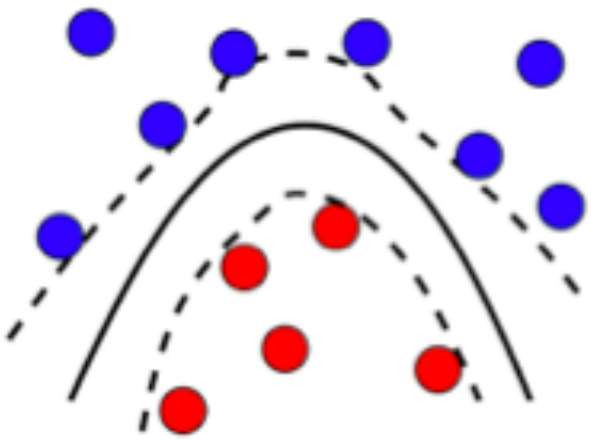
Example: SVM with a Gaussian kernel

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

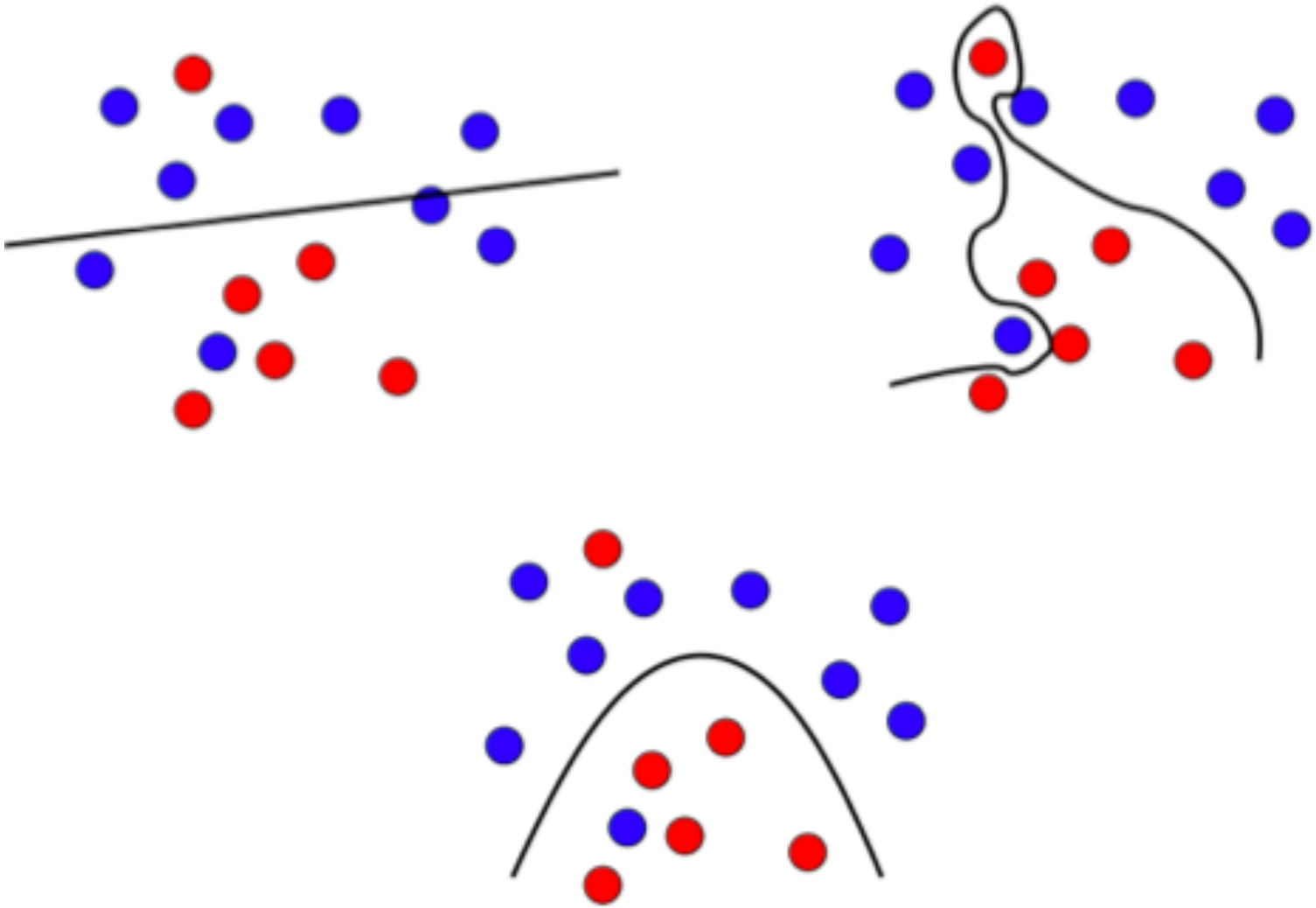


Linear vs nonlinear SVM





Regularity vs data fitting trade-off

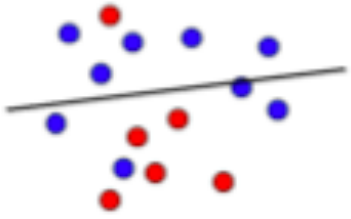


C controls the trade-off

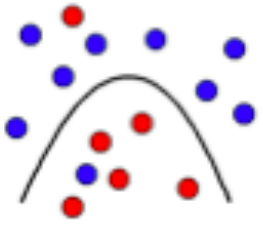
$$\min_f \left\{ \frac{1}{margin(f)} + C \times errors(f) \right\}$$

- Large C :
  - makes few errors

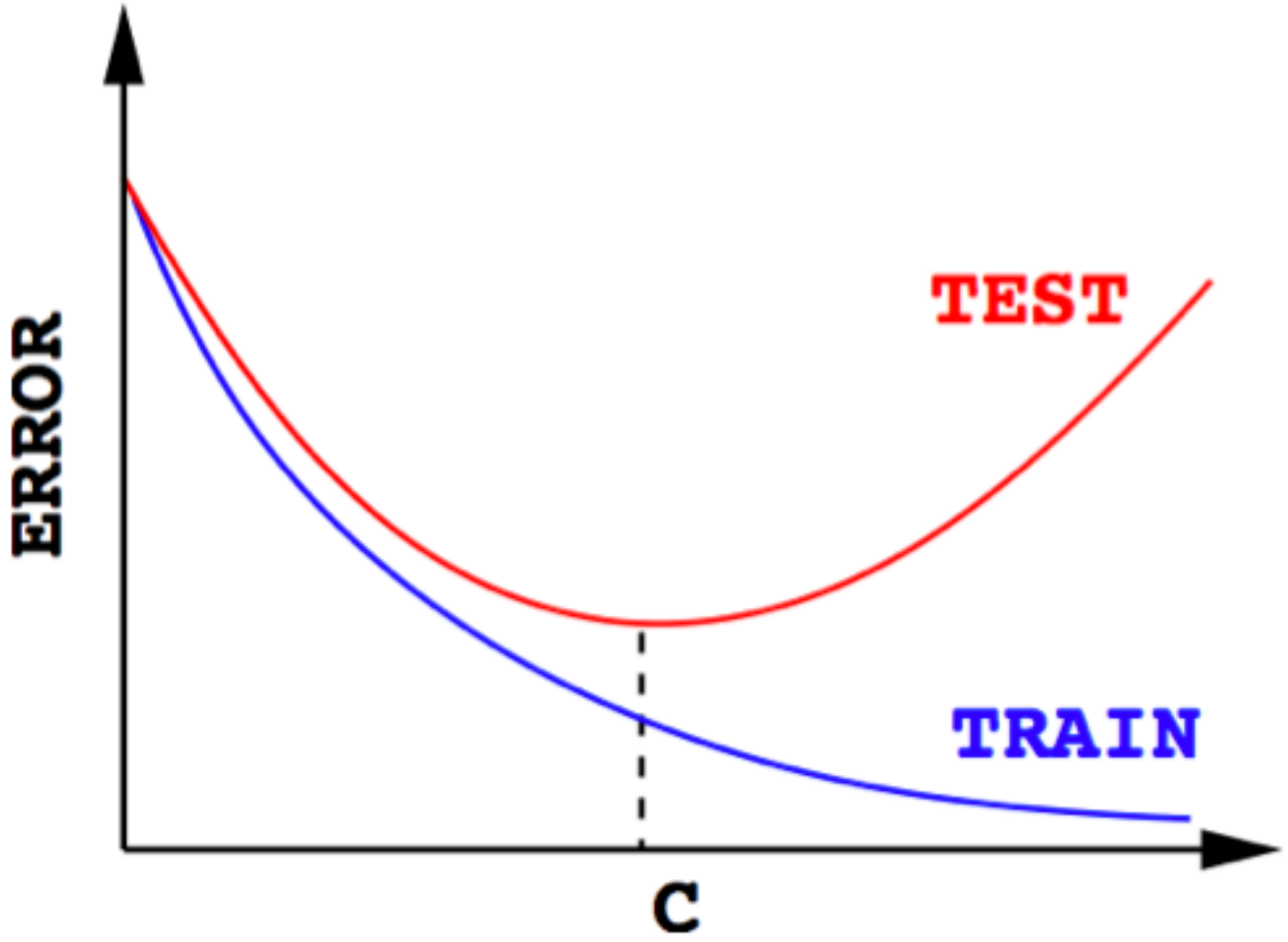
- Small C :
  - ensure a large margin



- Intermediate C:
  - finds a trade-off



Why it is important to control the trade-off

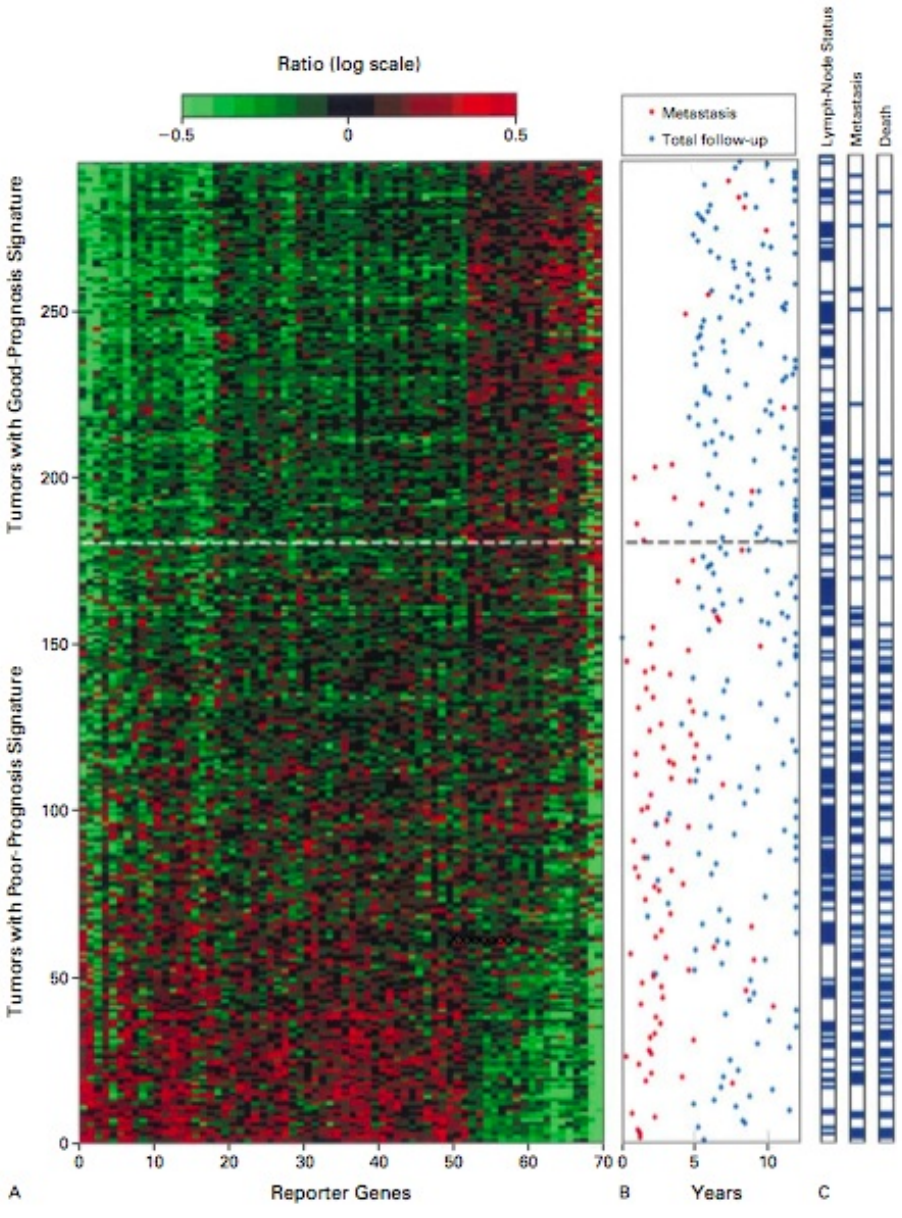


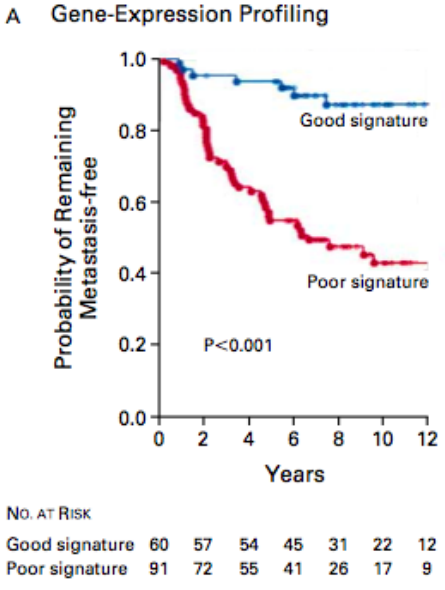
How to choose C in practice

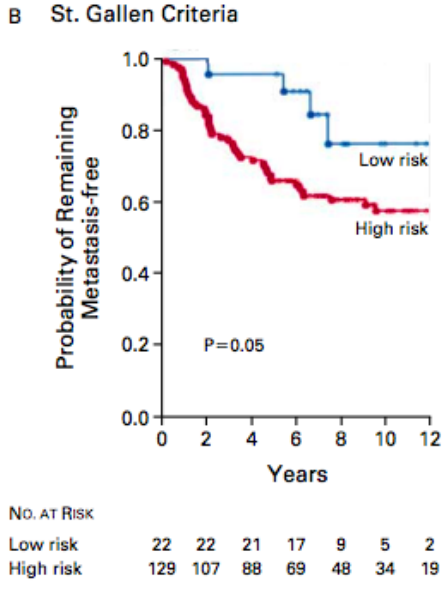
- Split your dataset in two ("train" and "test")
- Train SVM with different C on the "train" set
- Compute the accuracy of the SVM on the "test" set
- Choose the C which minimizes the "test" error
- (you may repeat this several times = cross-validation)

Outline

Breast cancer prognosis



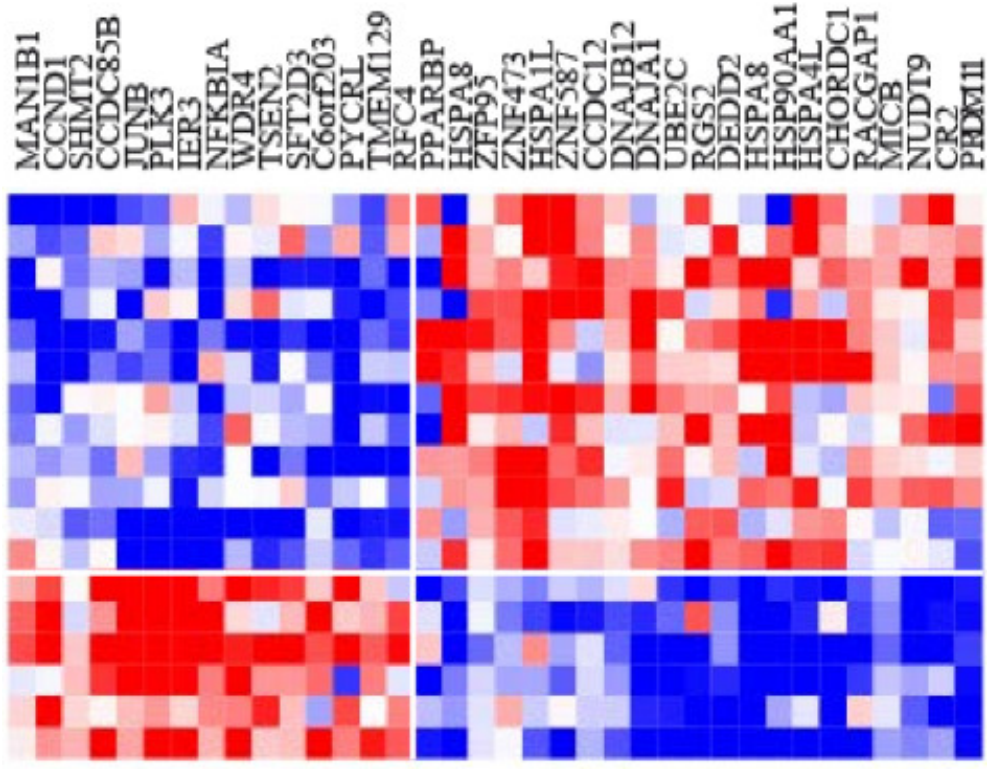




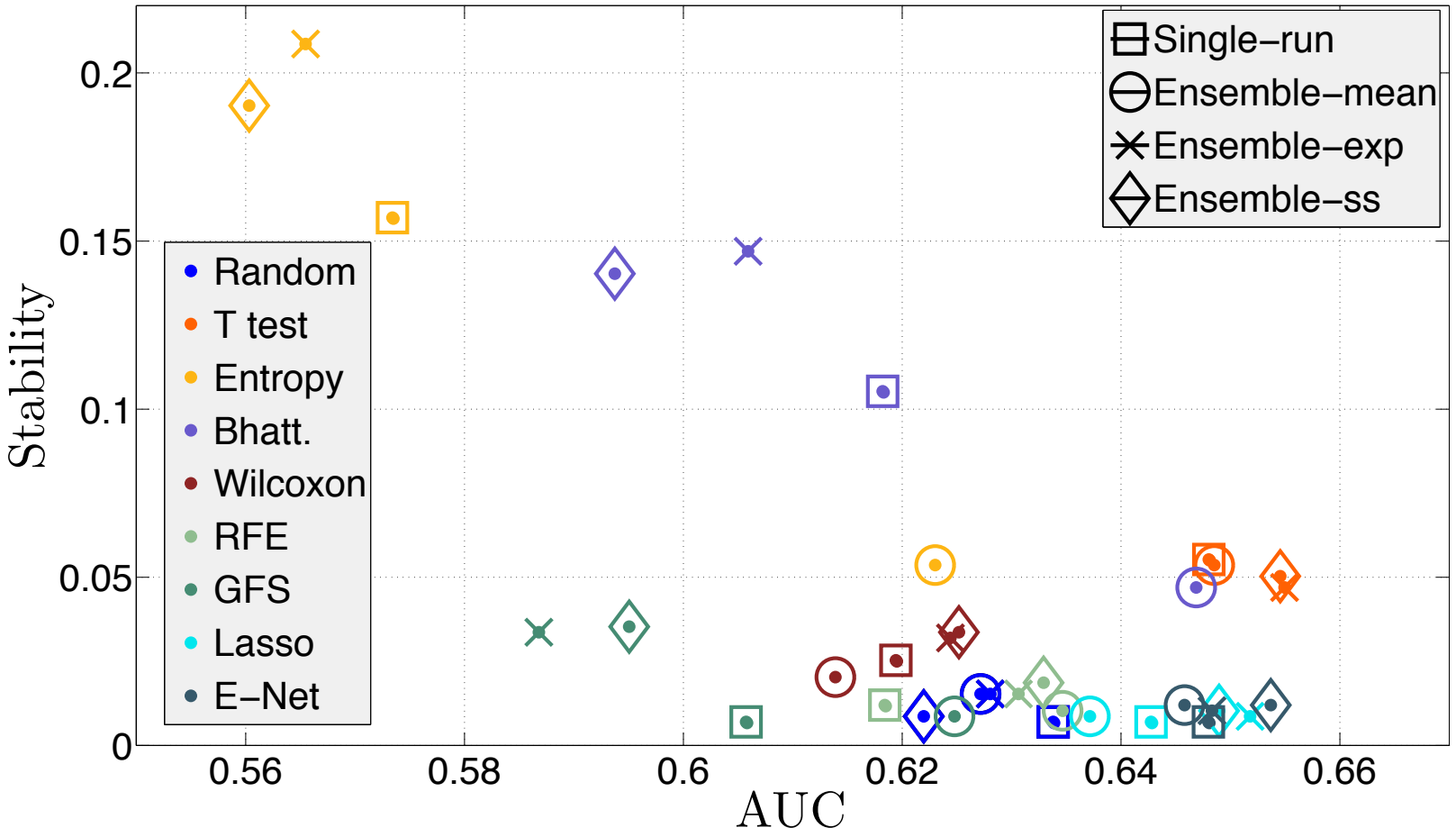
Gene selection, molecular signature

The idea

- We look for a limited set of genes that are sufficient for prediction.
- Selected genes should inform us about the underlying biology

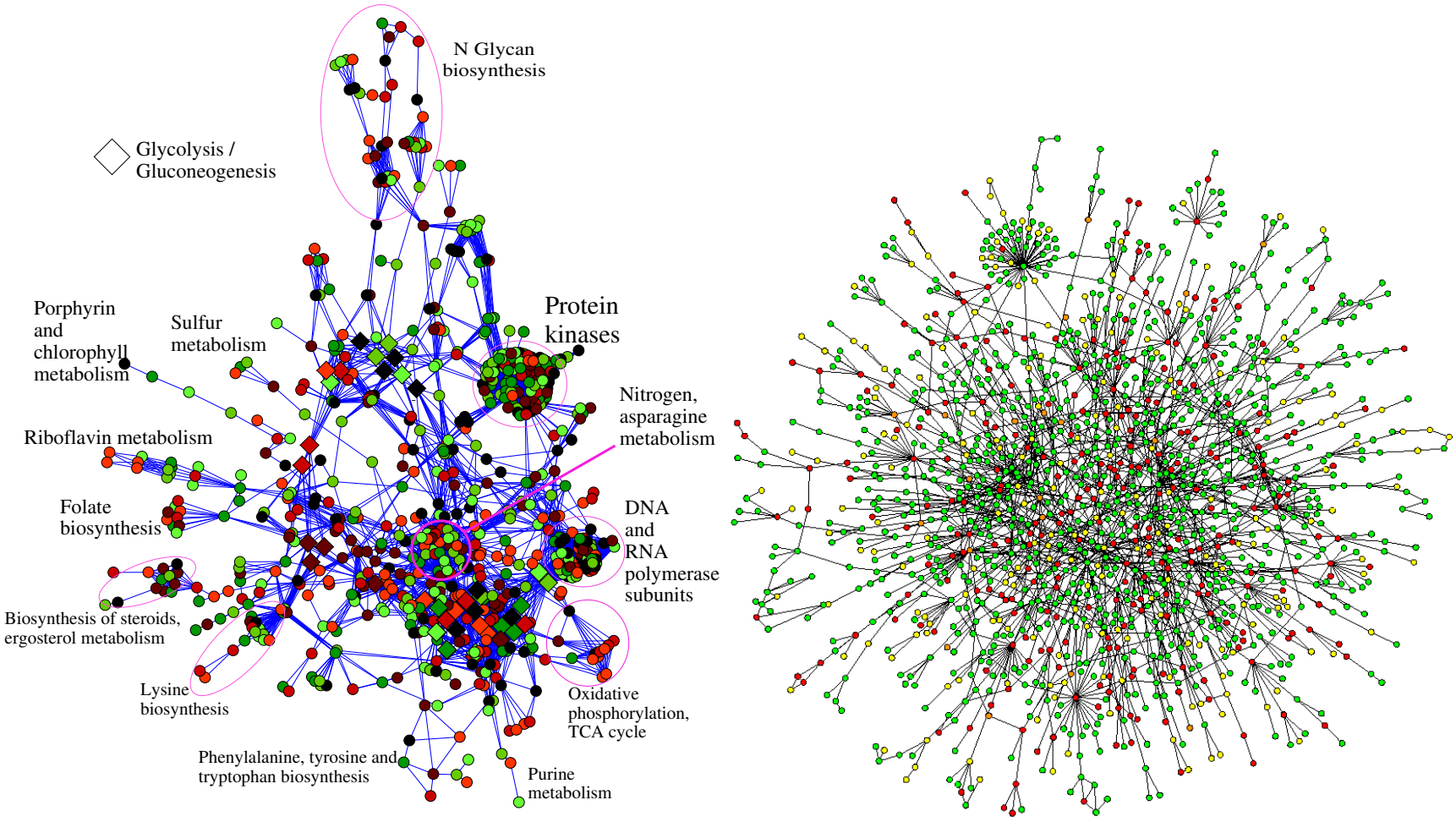


Lack of stability of signatures



Haury et al. (2011)

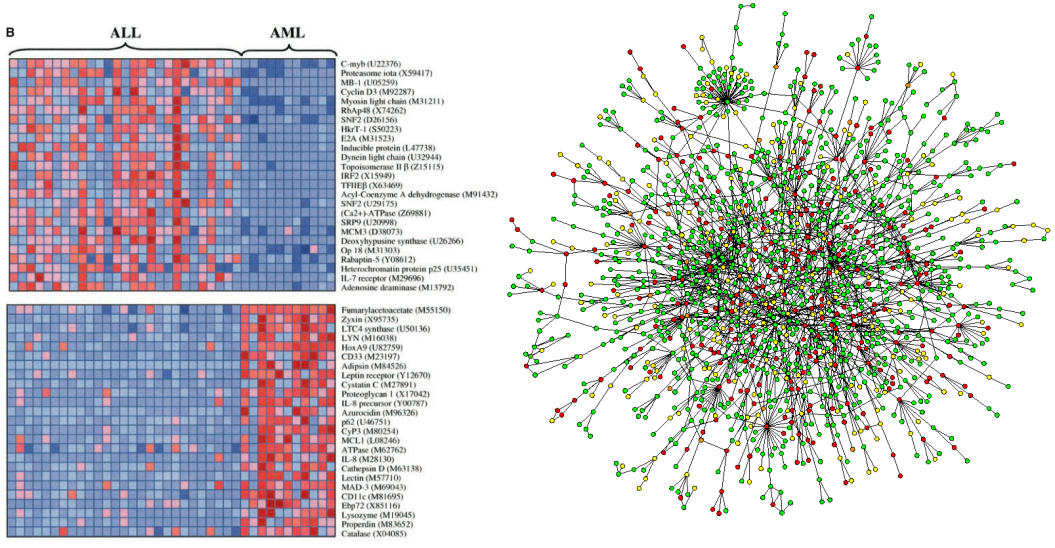
Gene networks



Gene networks and expression data

Motivation

- Basic biological functions usually involve the coordinated action of several proteins:
  - Formation of protein complexes
  - Activation of metabolic, signalling or regulatory pathways
- Many pathways and protein-protein interactions are already known
- Hypothesis: the weights of the classifier should be “coherent” with respect to this prior knowledge



Graph based penalty

$$f_\beta(x) = \beta^\top x \qquad \min_\beta R(f_\beta) + \lambda\Omega(\beta)$$

Prior hypothesis

Genes near each other on the graph should have similar weigths.

An idea (Rapaport et al., 2007)

$$\Omega(\beta) = \sum_{i\sim j}(\beta_i - \beta_j)^2 ,$$

$$\min_{\beta\in\mathbb{R}^p} R(f_\beta) + \lambda \sum_{i\sim j}(\beta_i - \beta_j)^2 .$$

Graph based penalty

$$f_\beta(x) = \beta^\top x \qquad \min_\beta R(f_\beta) + \lambda\Omega(\beta)$$

Prior hypothesis

Genes near each other on the graph should have similar weigths.

An idea (Rapaport et al., 2007)

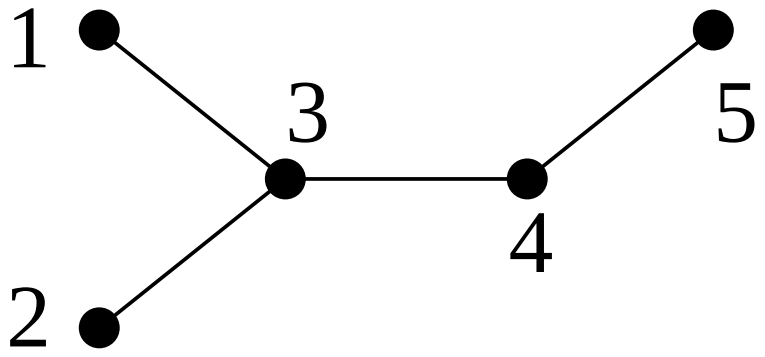
$$\Omega(\beta) = \sum_{i\sim j}(\beta_i - \beta_j)^2 \,,$$

$$\min_{\beta\in\mathbb{R}^p} R(f_\beta) + \lambda \sum_{i\sim j}(\beta_i - \beta_j)^2 \,.$$

Graph Laplacian

Definition

The Laplacian of the graph is the matrix $L = D - A$.



$$L = D - A = \begin{pmatrix} 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Spectral penalty as a kernel

Theorem

The function $f(x) = \beta^\top x$ where β is solution of

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} \ell\left(\beta^\top x_i, y_i\right) + \lambda \sum_{i \sim j} \left(\beta_i - \beta_j\right)^2$$

is equal to $g(x) = \gamma^\top \Phi(x)$ where γ is solution of

$$\min_{\gamma \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} \ell\left(\gamma^\top \Phi(x_i), y_i\right) + \lambda \gamma^\top \gamma \,,$$

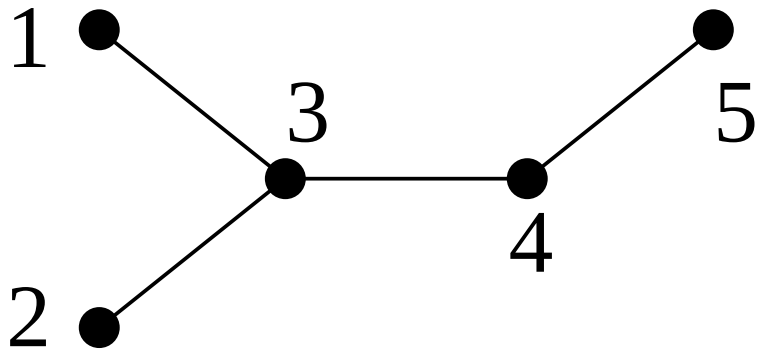
and where

$$\Phi(x)^\top \Phi(x') = x^\top K_G x'$$

for $K_G = L^*$, the pseudo-inverse of the graph Laplacian.

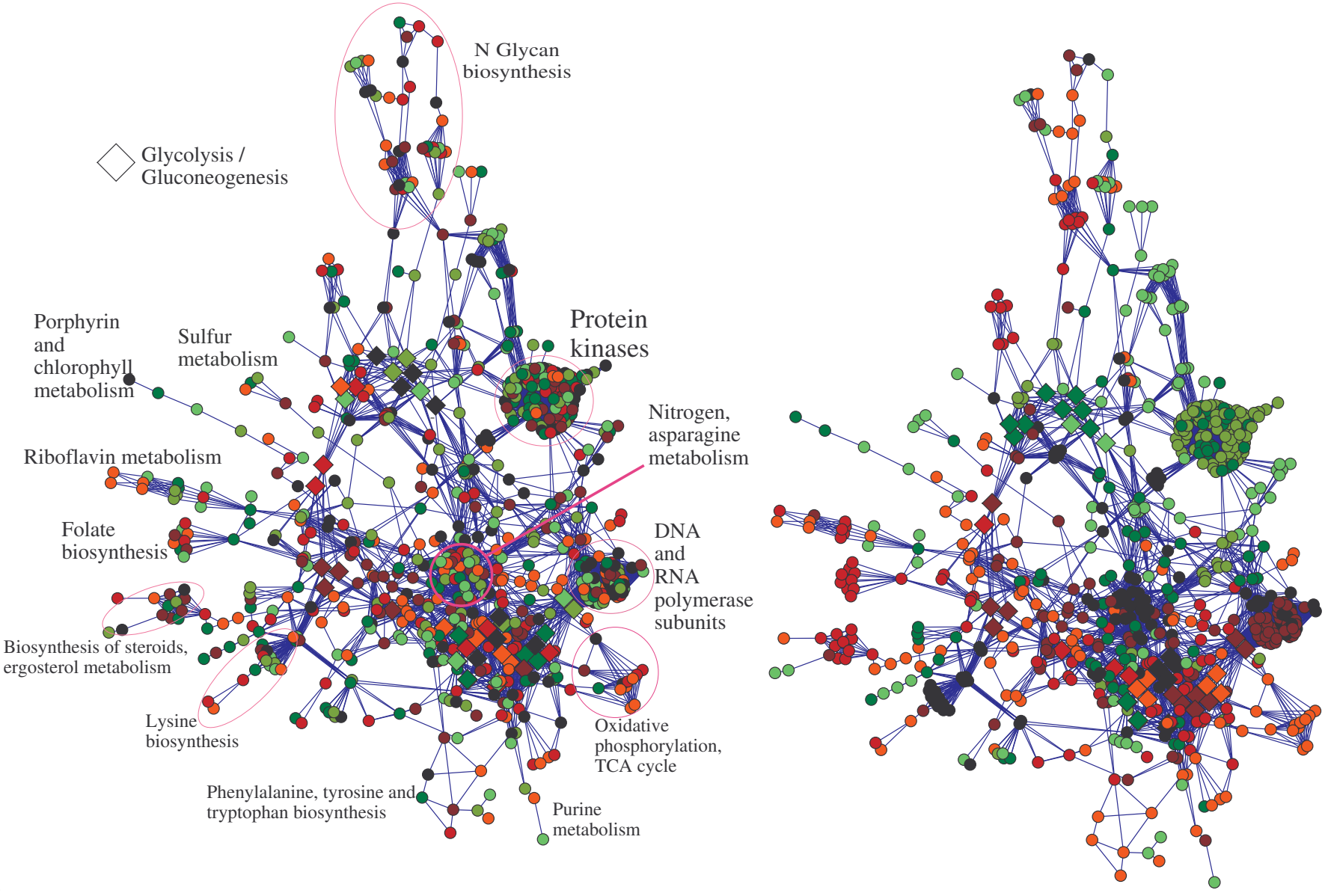
Proof: left as exercice

Example

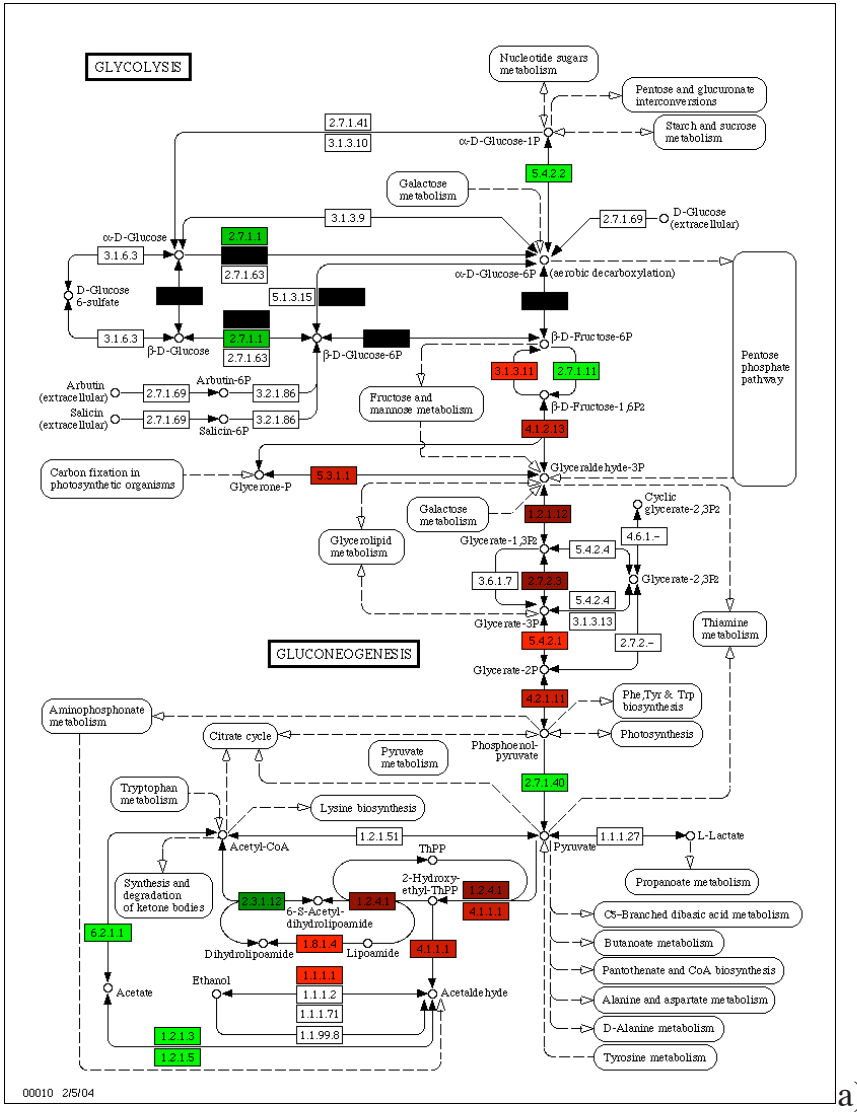


$$L^* = \begin{pmatrix} 0.88 & -0.12 & 0.08 & -0.32 & -0.52 \\ -0.12 & 0.88 & 0.08 & -0.32 & -0.52 \\ 0.08 & 0.08 & 0.28 & -0.12 & -0.32 \\ -0.32 & -0.32 & -0.12 & 0.48 & 0.28 \\ -0.52 & -0.52 & -0.32 & 0.28 & 1.08 \end{pmatrix}$$

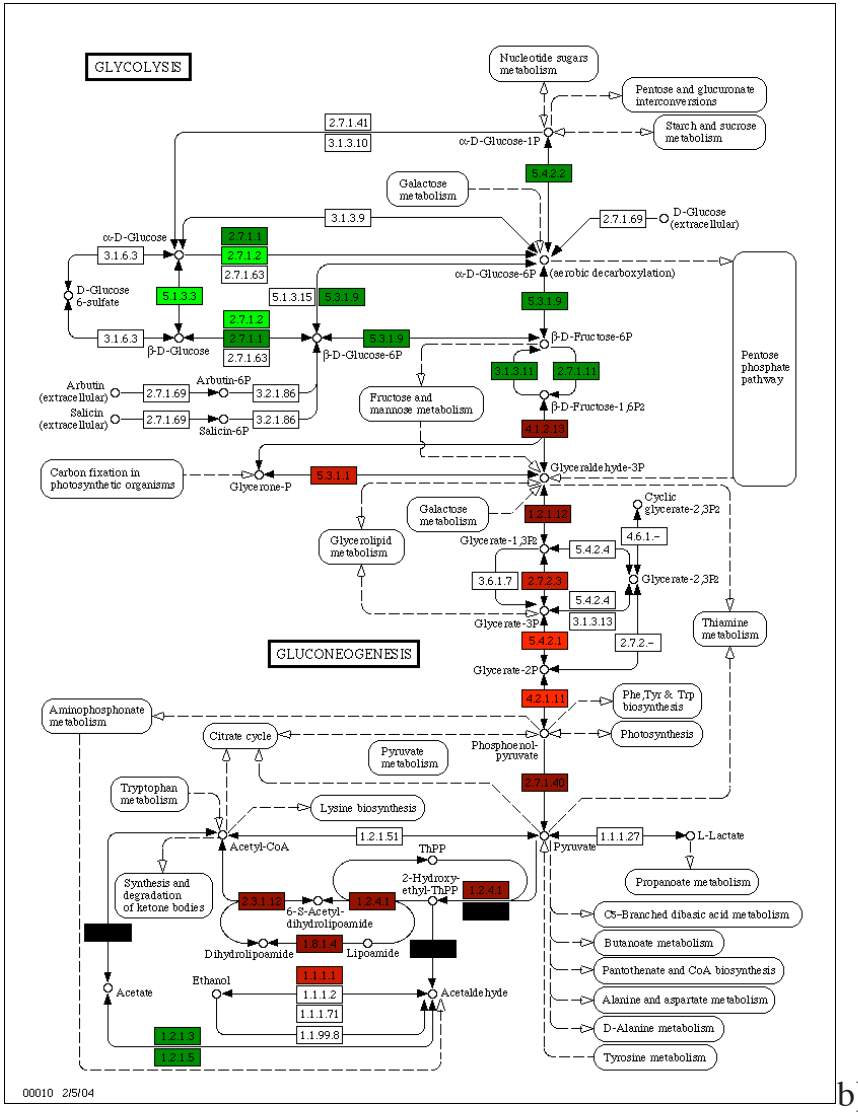
Classifiers



Classifier



a)



b)

Other penalties with kernels

$$\Phi(x)^\top \Phi(x') = x^\top K_G x'$$

with:

- $K_G = (c + L)^{-}1$ leads to

$$\Omega(\beta) = c \sum_{i=1}^{p} \beta_i^2 + \sum_{i \sim j} (\beta_i - \beta_j)^2 \ .$$

- The diffusion kernel:

$$K_G = \exp_M(-2tL) \, .$$

penalizes high frequencies of β in the Fourier domain.

Outline

Supervised sequence classification

Data (training)

- Secreted proteins:

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

- Non-secreted proteins:

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

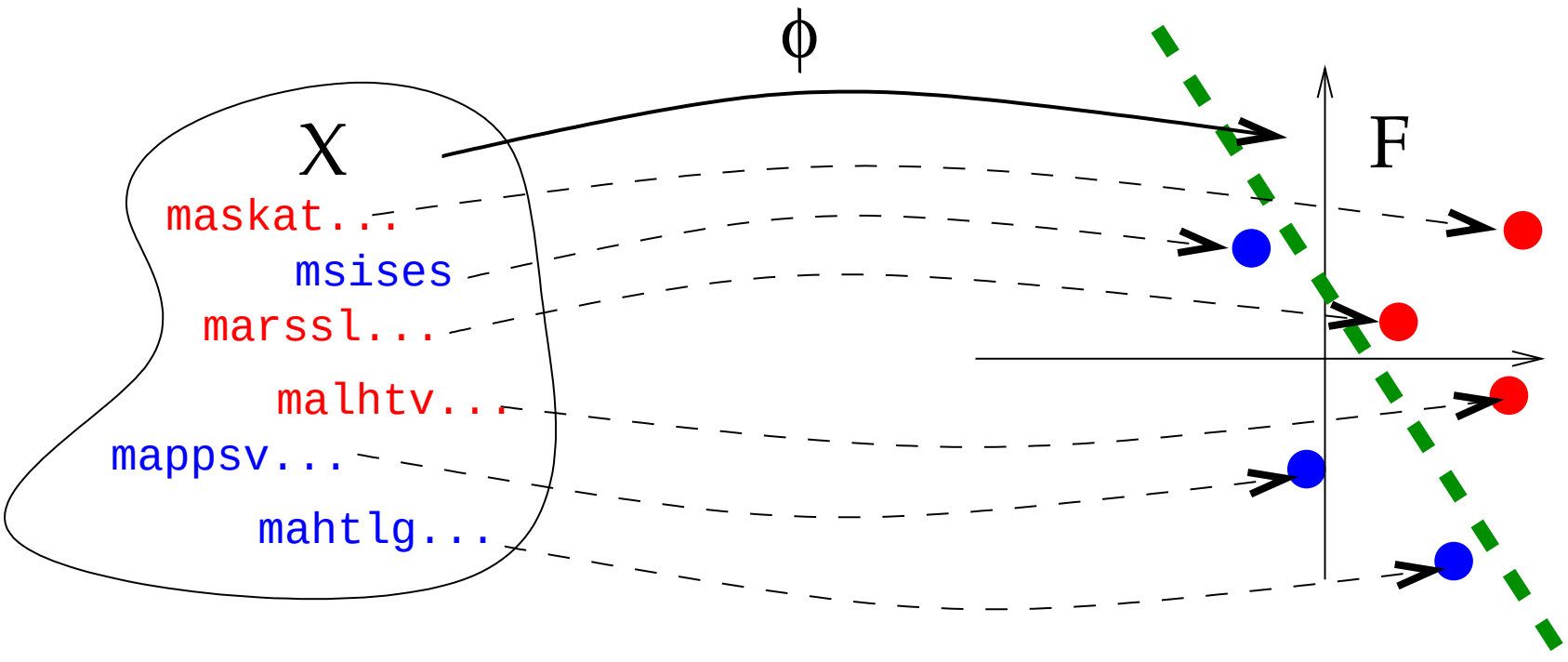
Goal

- 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:

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

- 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:

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

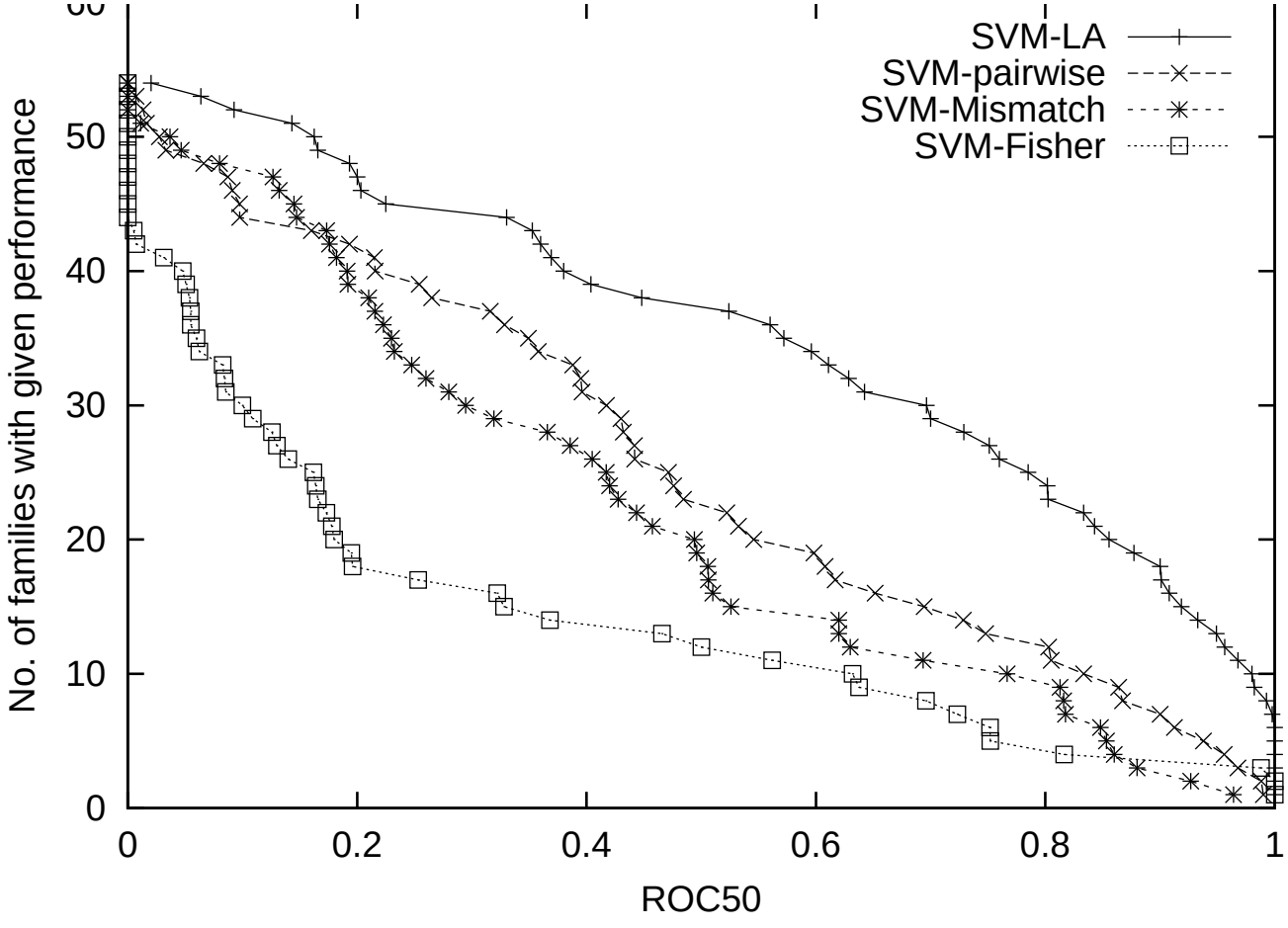
- A kernel for letters:

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

- A kernel for gaps:

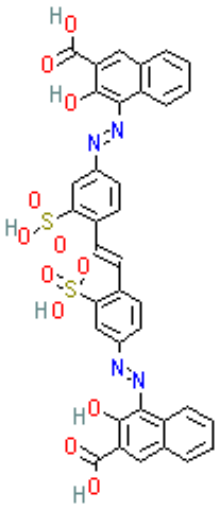
$$K_g^{(\beta)} (\mathbf{x}, \mathbf{y}) = \exp\left[\beta \left(g(|\mathbf{x}|) + g(|\mathbf{x}|) \right)\right] \,.$$

The choice of kernel matters

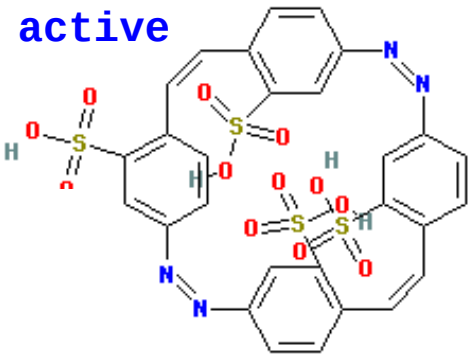


Performance on the SCOP superfamily recognition benchmark (from Saigo et al., 2004).

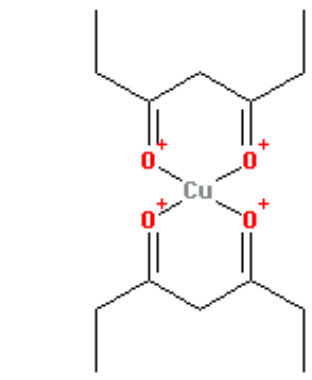
Virtual screening for drug discovery

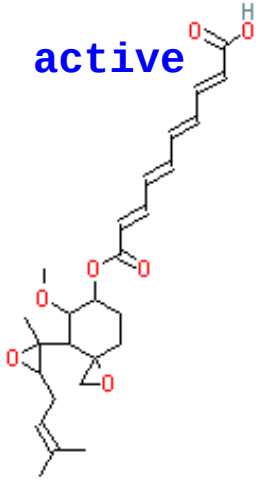


active

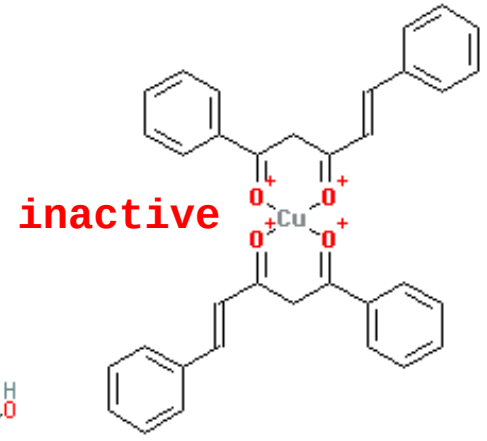


inactive

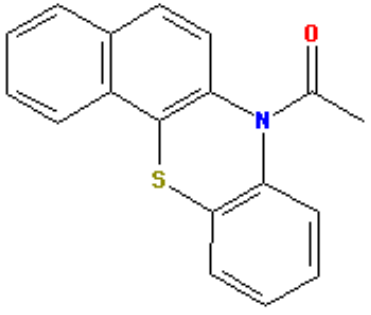




active



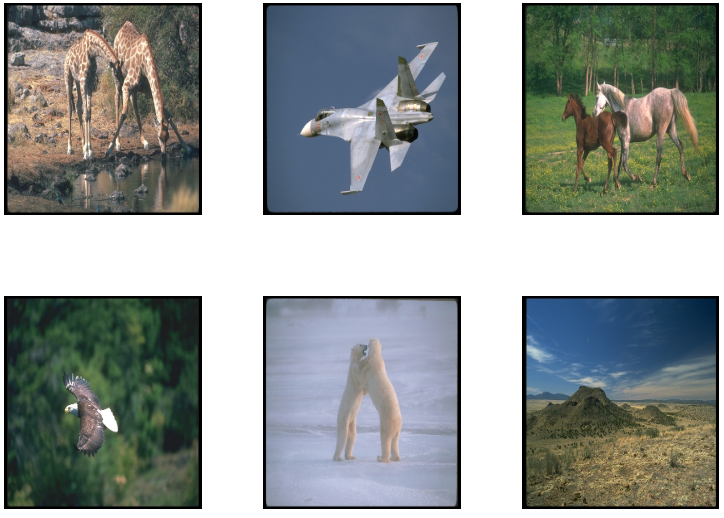
inactive

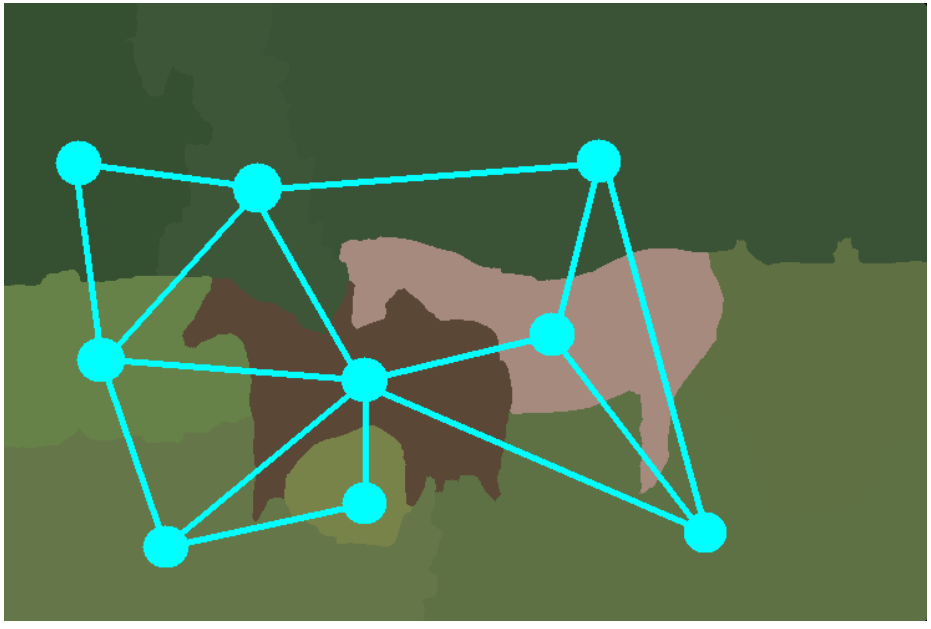


inactive

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

Image retrieval and classification





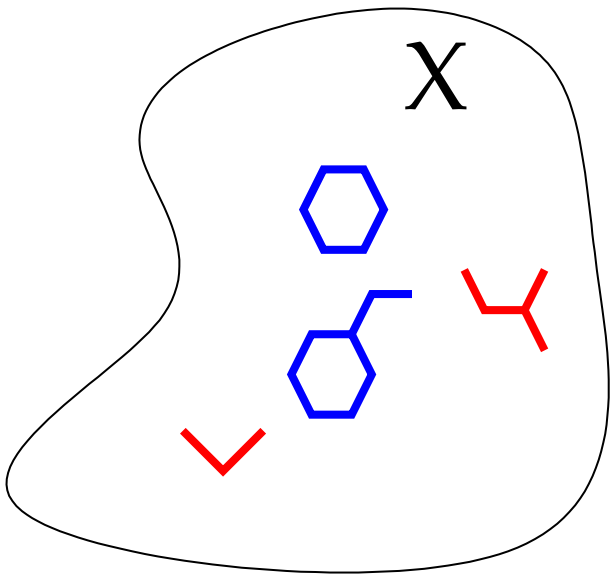
From Harchaoui and Bach (2007).

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 $\mathcal{H}$.

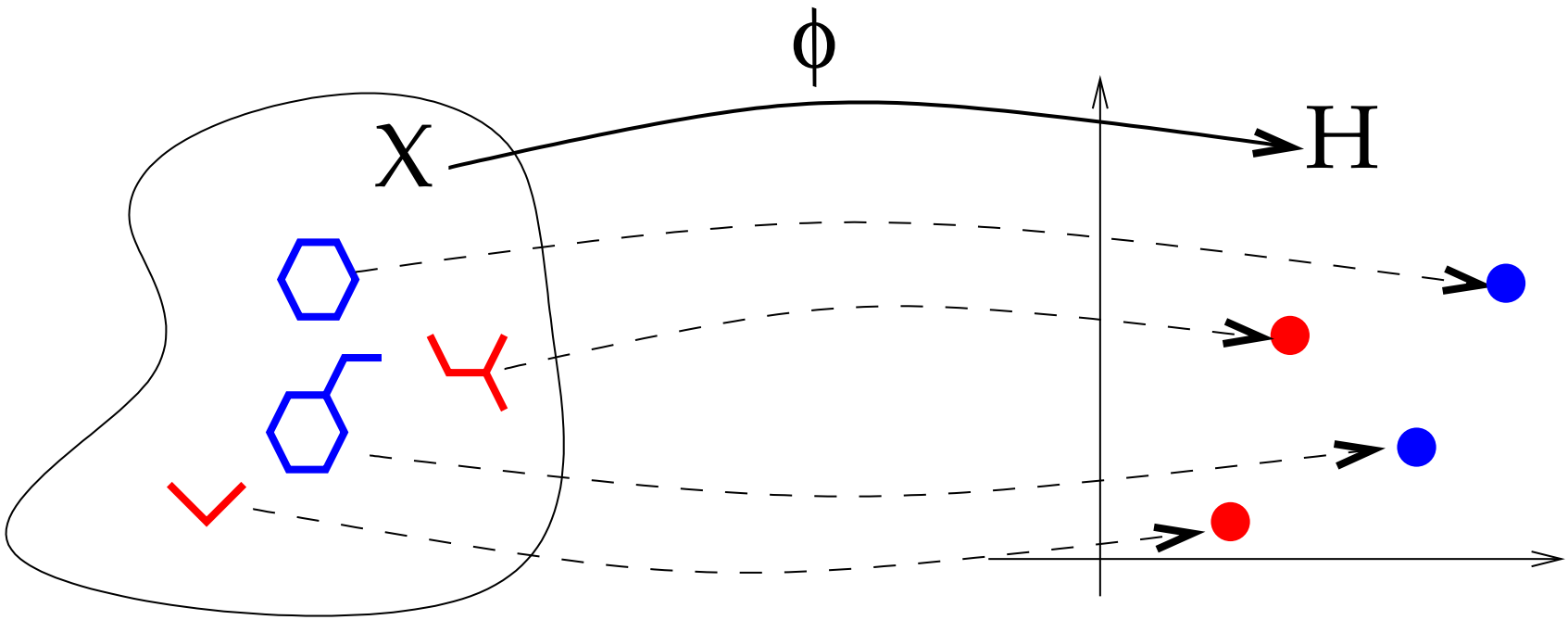


Graph kernels

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 $\mathcal{H}$.

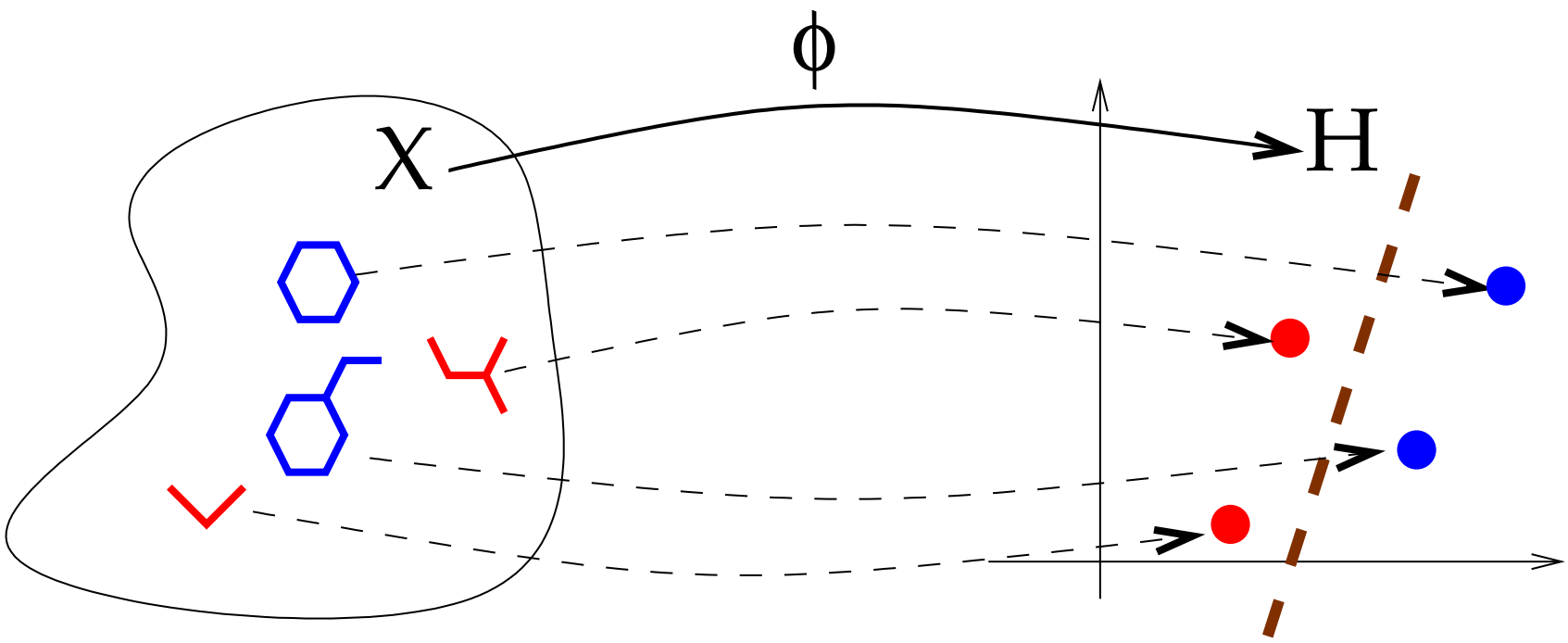


Graph kernels

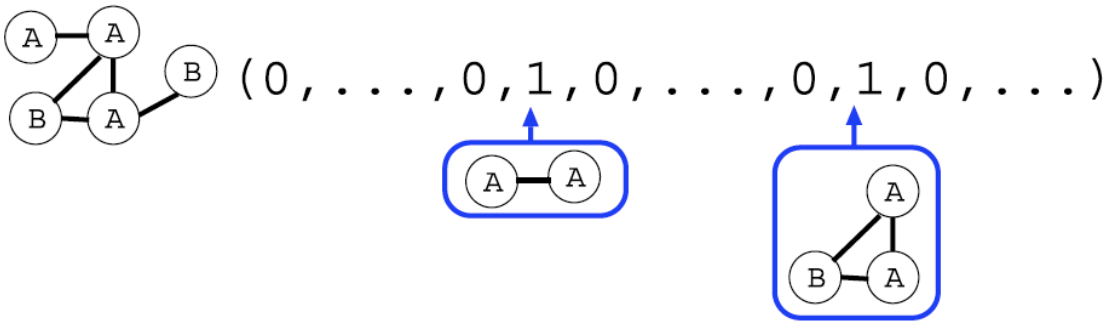
1. Represent each graph x by a vector $\Phi(x) \in \mathcal{H}$, either explicitly or implicitly through the kernel

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

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



Indexing by all subgraphs?



Theorem

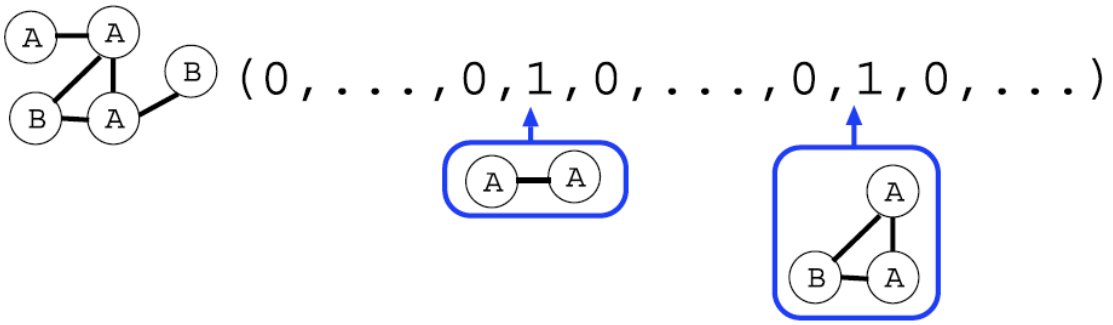
Computing all subgraph occurrences is NP-hard.

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

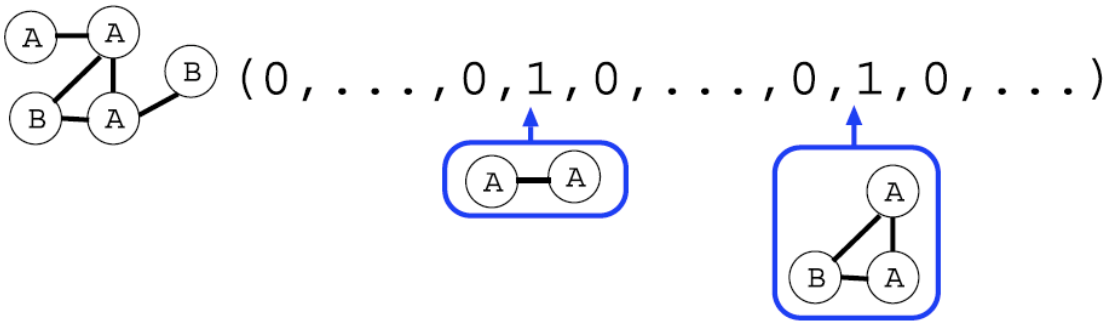
Computing all subgraph occurrences is NP-hard.

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.

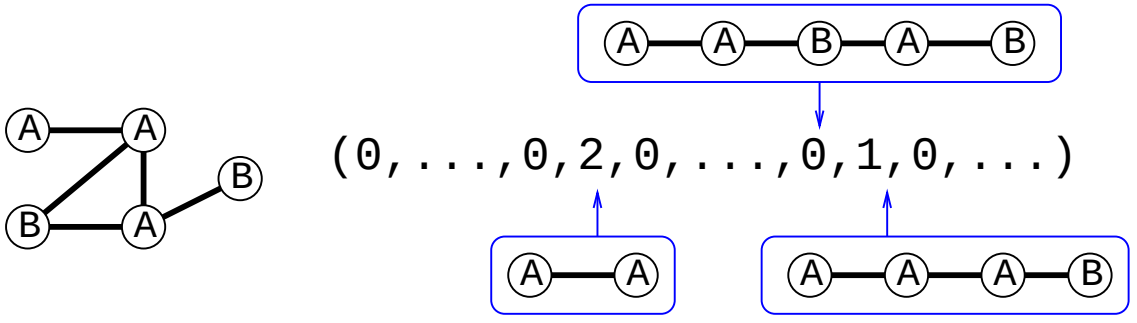
Indexing by specific subgraphs

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)

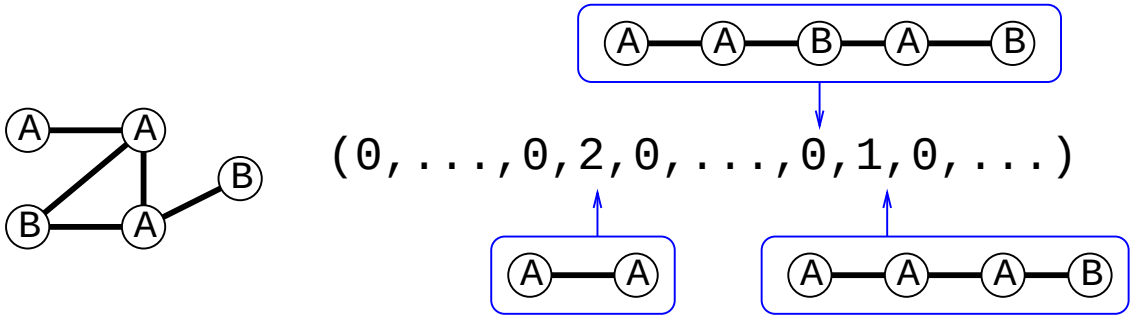
Example : Indexing by all shortest paths



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.

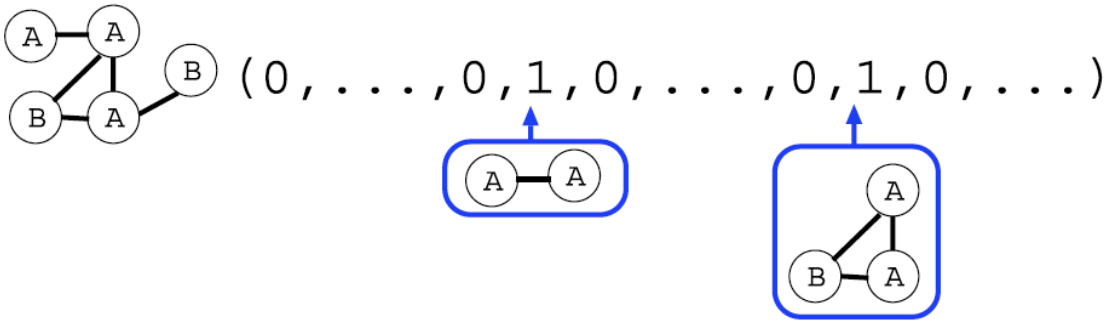
Example : Indexing by all shortest paths



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.

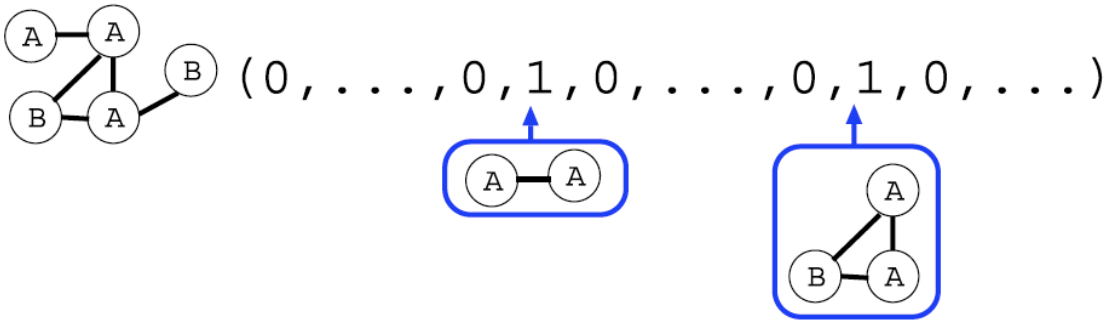
Example : Indexing by all subgraphs up to k vertices



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



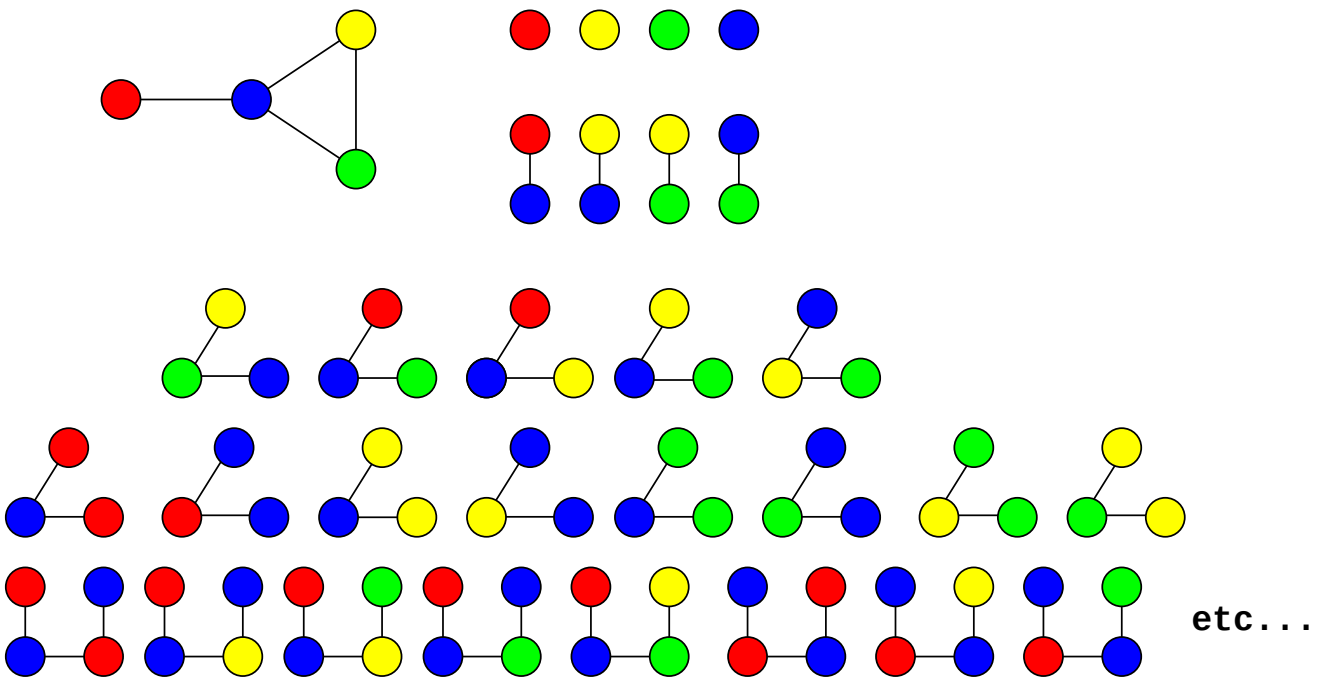
Properties (Shervashidze et al., 2009)

- 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.

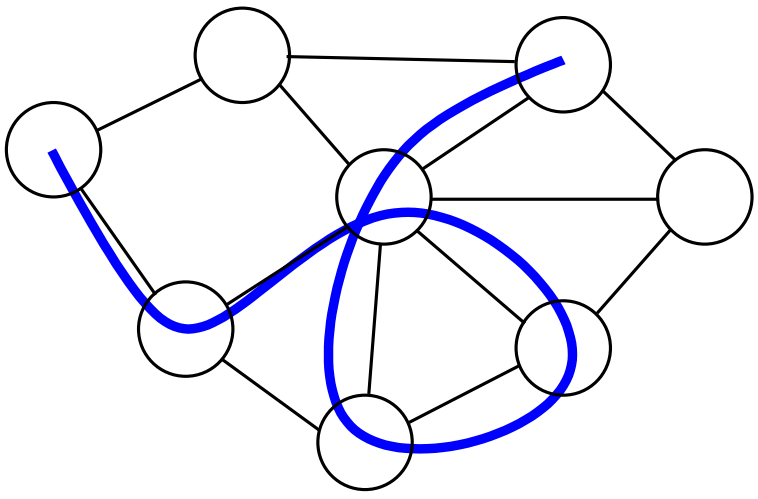
Walks

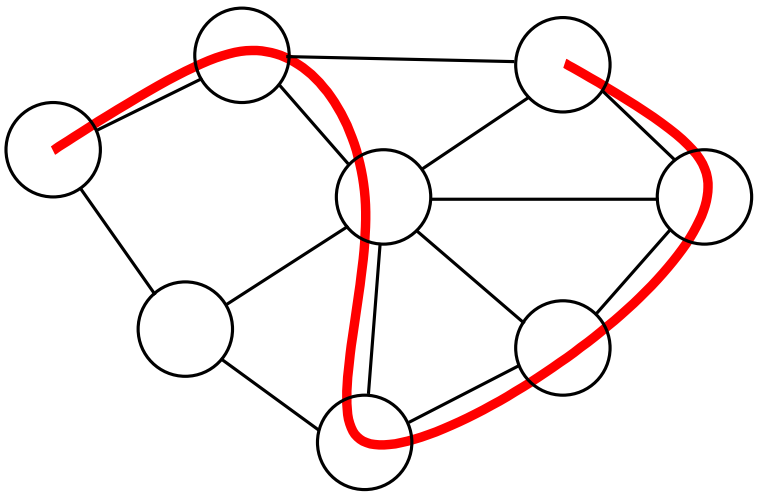
Definition

- 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

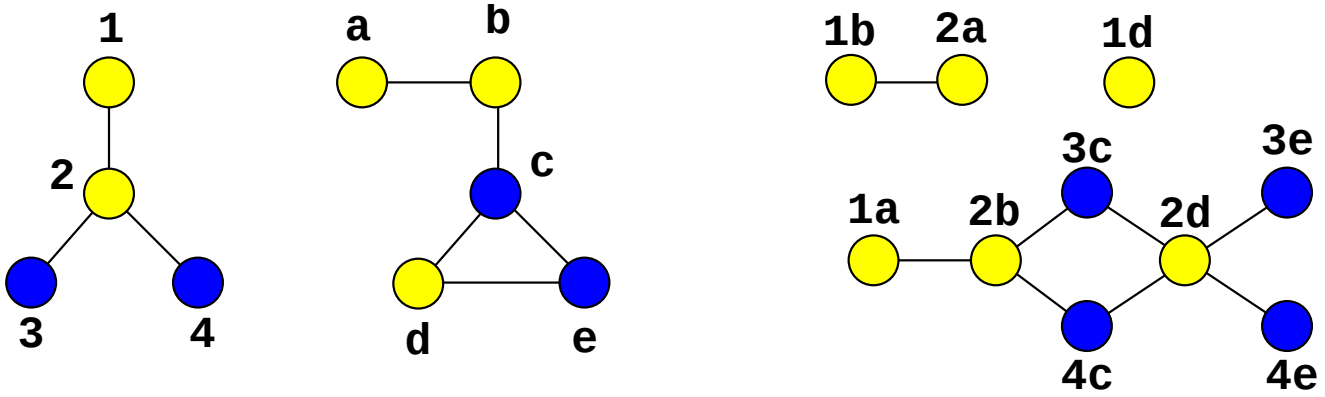
These three kernels (*n*th-order, random and geometric walk kernels) can be computed efficiently in polynomial time.

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:

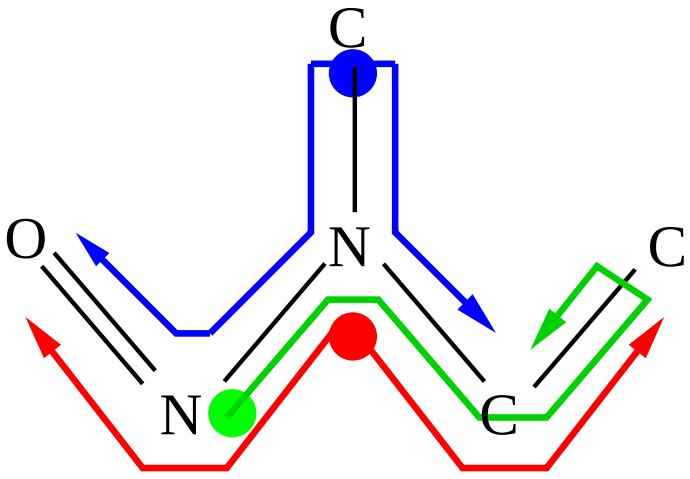
$$\lambda_G(v_1 \dots v_n) = \lambda^i(v_1) \prod_{i=2}^{n} \lambda^t(v_{i-1}, v_i)\,.$$

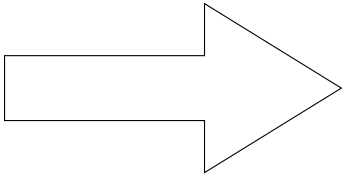
- Let Λ_i be the vector of $\lambda^i(v)$ and Λ_t be the matrix of $\lambda^t(v, v')$:

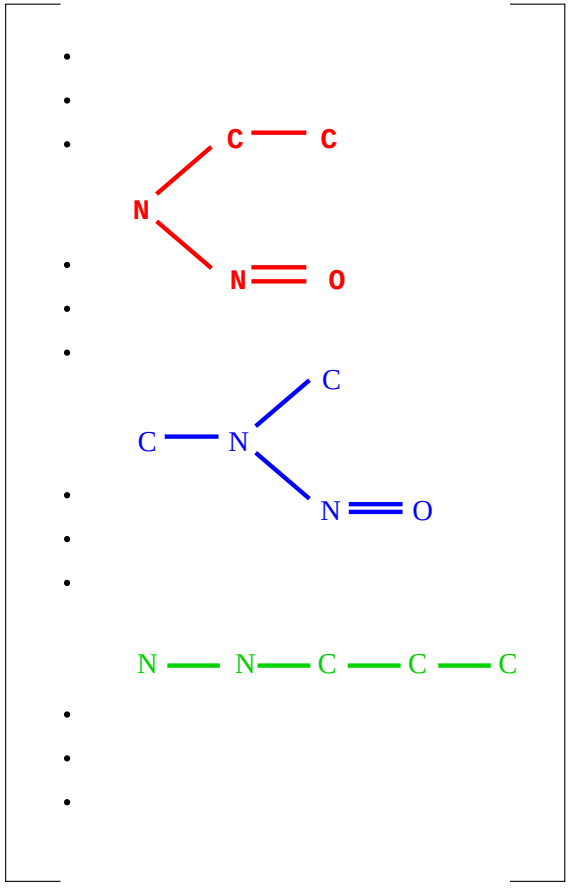
$$\begin{aligned} K_{walk}(G_1, G_2) &= \sum_{n=1}^{\infty} \sum_{w \in \mathcal{W}_n(G_1 \times G_2)} \lambda^i(v_1) \prod_{i=2}^{n} \lambda^t(v_{i-1}, v_i) \\ &= \sum_{n=0}^{\infty} \Lambda_i \Lambda_t^n \mathbf{1} \\ &= \Lambda_i \left(I - \Lambda_t\right)^{-1} \mathbf{1} \end{aligned}$$

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

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

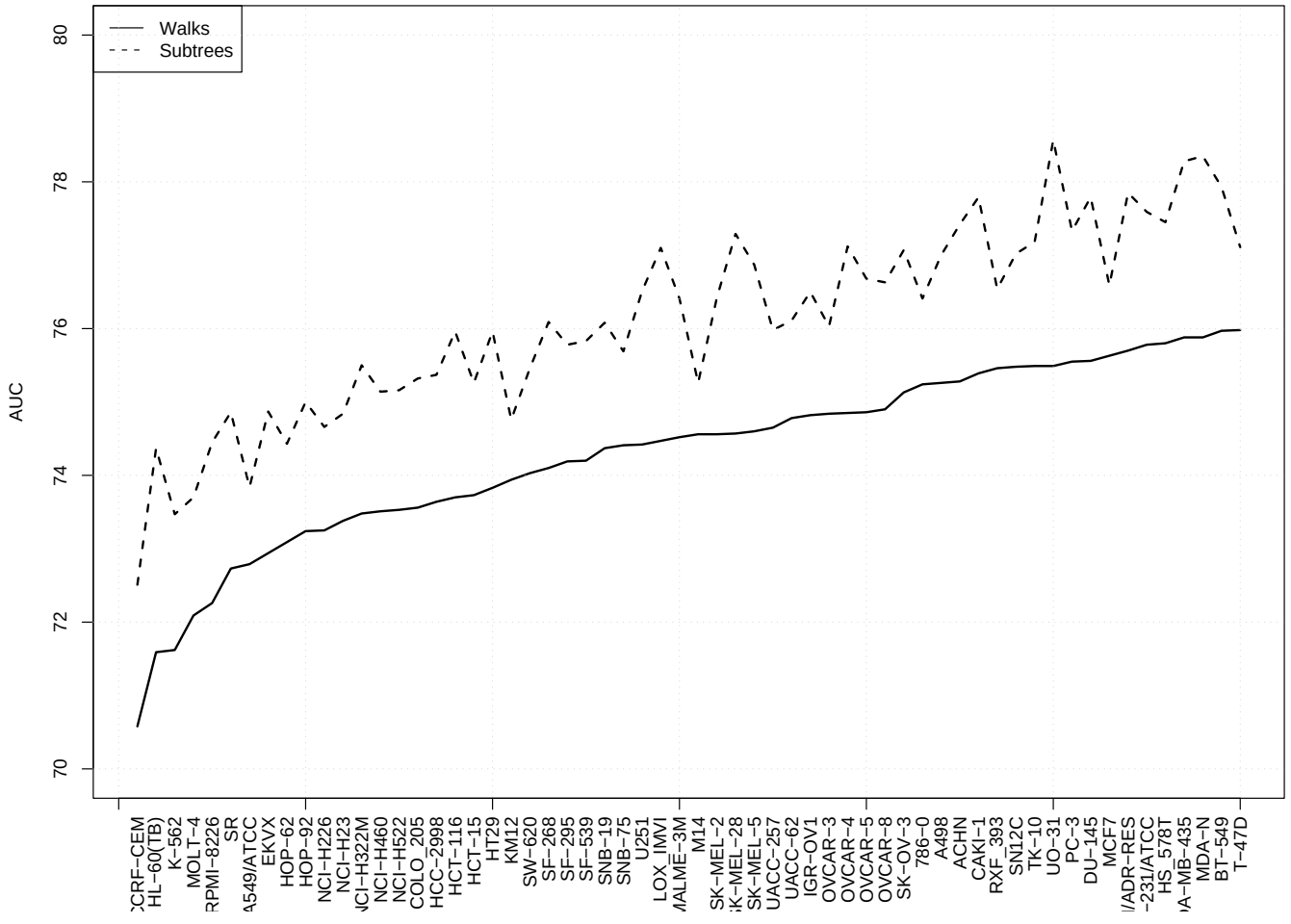






$$\mathcal{T}(v, n+1) = \sum_{R \subset \mathcal{N}(v)} \prod_{v' \in R} \lambda_t(v, v') \mathcal{T}(v', n)\,,$$

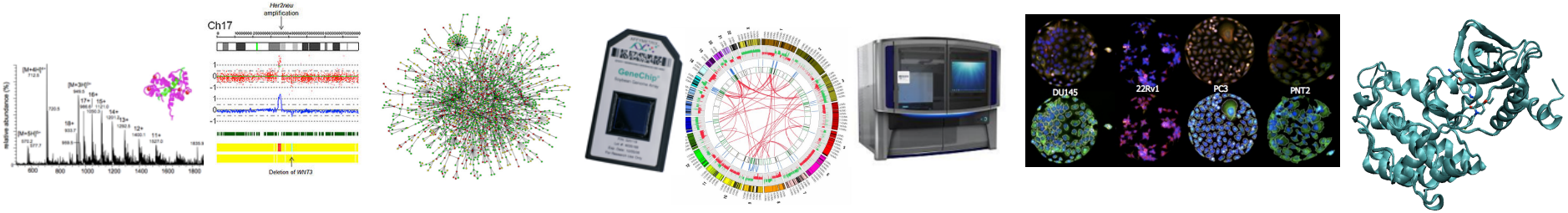
2D Subtree vs walk kernels



Screening of inhibitors for 60 cancer cell lines.

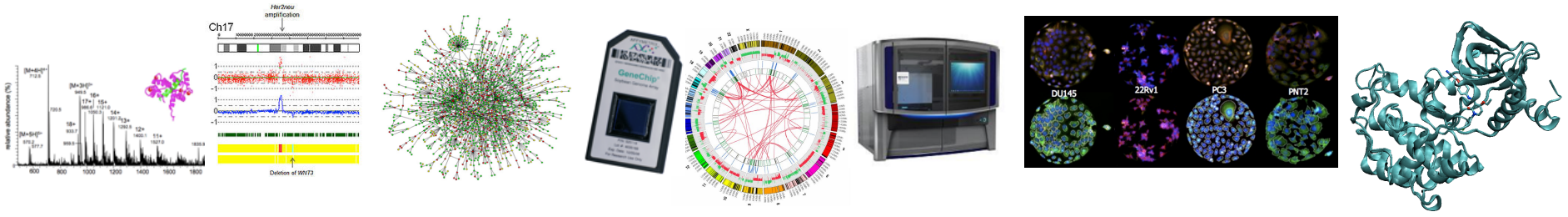
Outline

Motivation



- Assume we observe M types of data and would like to learn a joint model (e.g., predict susceptibility from SNP and expression data).
- We saw in the previous part how to make kernels $K_1, \ldots, K_M$ for each type of data, and learn with each kernel individually
- Can we combine them to learn jointly from heterogeneous data?

Sum kernel



Definition

Let $K_1, \ldots, K_M$ be M kernels on $\mathcal{X}$. The sum kernel K_S is the kernel on $\mathcal{X}$ defined as

$$\forall x, x' \in \mathcal{X}\,, \quad K_S(x, x') = \sum_{i=1}^{M} K_i(x, x')\,.$$

Sum kernel and vector concatenation

Theorem

For $i = 1, \ldots, M$, let $\Phi_i : \mathcal{X} \to \mathcal{H}_i$ be a feature map such that

$$K_i(x, x') = \left\langle \Phi_i(x), \Phi_i\left(x'\right) \right\rangle_{\mathcal{H}_i} .$$

Then $K_S = \sum_{i=1}^{M} K_i$ can be written as:

$$K_S(x, x') = \left\langle \Phi_S(x), \Phi_S\left(x'\right) \right\rangle_{\mathcal{H}_S} ,$$

where $\Phi_S : \mathcal{X} \to \mathcal{H}_S = \mathcal{H}_1 \oplus \ldots \oplus \mathcal{H}_M$ is the concatenation of the feature maps Φ_i:

$$\Phi_S(x) = \left(\Phi_1(x), \ldots, \Phi_M(x)\right)^\top .$$

Therefore, summing kernels amounts to concatenating their feature space representations, which is a quite natural way to integrate different features.

Proof

For $\Phi_S(x) = (\Phi_1(x), \ldots, \Phi_M(x))^\top$, we easily compute:

$$\begin{aligned}\left\langle \Phi_S(x), \Phi_S(x') \right\rangle_{\mathcal{H}_s} &= \sum_{i=1}^{M} \left\langle \Phi_i(x), \Phi_i(x') \right\rangle_{\mathcal{H}_i} \\ &= \sum_{i=1}^{M} K_i(x, x') \\ &= K_S(x, x').\end{aligned}$$

Example: data integration with the sum kernel

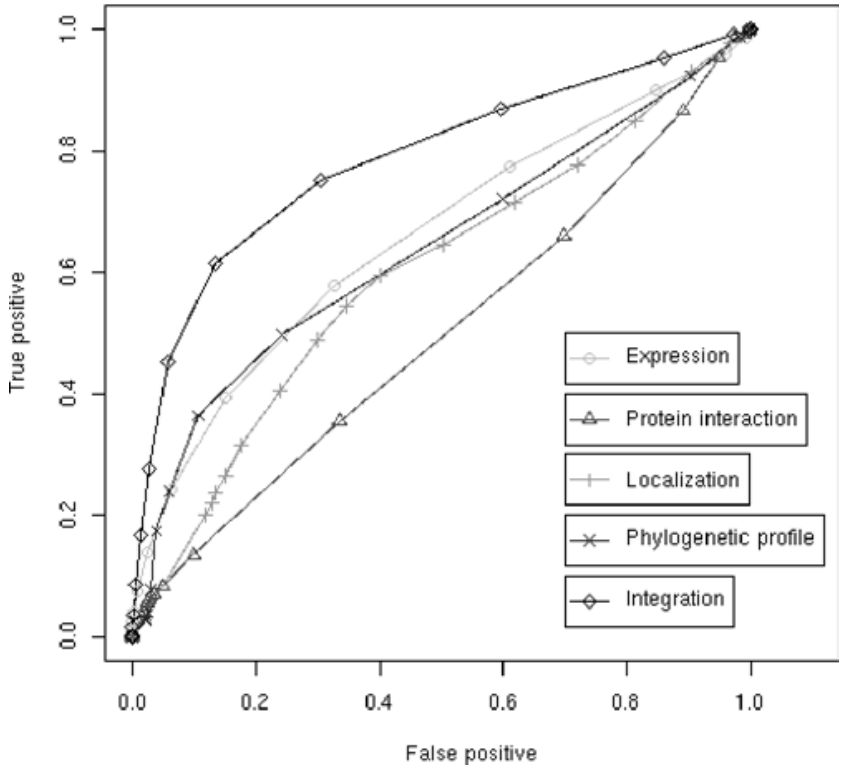
BIOINFORMATICS

Vol. 20 Suppl. 1 2004, pages i363–i370
DOI: 10.1093/bioinformatics/bth910

Protein network inference from multiple genomic data: a supervised approach

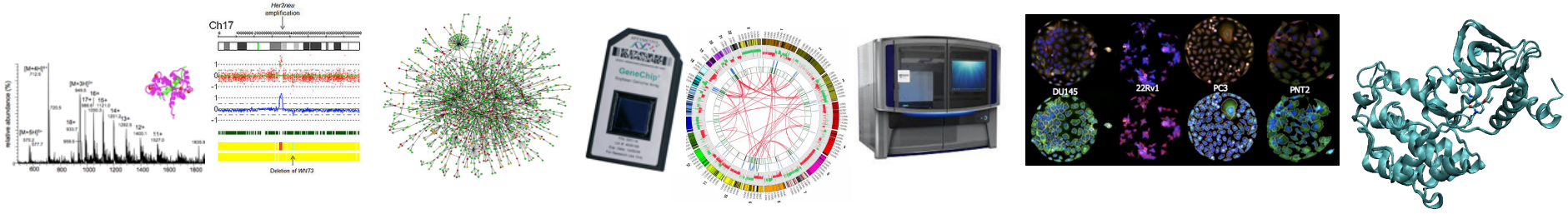
Y. Yamanishi[1,], J.-P. Vert[2] and M. Kanehisa[1]*

[1]Bioinformatics Center, Institute for Chemical Research, Kyoto University, Gokasho, Uji, Kyoto 611-0011, Japan and [2]Computational Biology group, Ecole des Mines de Paris, 35 rue Saint-Honoré, 77305 Fontainebleau cedex, France



K_{exp} (Expression)
K_{ppi} (Protein interaction)
K_{loc} (Localization)
K_{phy} (Phylogenetic profile)
$K_{\text{exp}} + K_{\text{ppi}} + K_{\text{loc}} + K_{\text{phy}}$ (Integration)

Learning the kernel



Motivation

- If we know how to weight each kernel, then we can learn with the weighted kernel

$$K_\eta = \sum_{i=1}^{M} \eta_i K_i$$

- However, usually we don't know...
- Perhaps we can optimize the weights η_i during learning?

An objective function for K

Theorem

For any p.d. kernel K on $\mathcal{X}$, let

$$J(K) = \min_{f \in \mathcal{H}_K} \left\{ R(f^n) + \lambda \| \beta \|^2_{\mathcal{H}_K} \right\} .$$

The function $K \mapsto J(K)$ is convex.

This suggests a principled way to "learn" a kernel: define a convex set of candidate kernels, and minimize $J(K)$ by convex optimization.

Proof

- We can show by strong duality that

$$J(K) = \max_{\gamma \in \mathbb{R}^n} \left\{ -R^*(-2\lambda\gamma) - \lambda\gamma^\top K\gamma \right\} .$$

- For each γ fixed, this is an affine function of K, hence convex
- A supremum of convex functions is convex. $\square$

MKL (Lanckriet et al., 2004)

- We consider the set of convex combinations

$$K_\eta = \sum_{i=1}^{M} \eta_i K_i \quad \text{with} \quad \eta \in \Sigma_M = \left\{ \eta_i \geq 0 \,, \sum_{i=1}^{M} \eta_i = 1 \right\}$$

- We optimize both η and f^* by solving:

$$\min_{\eta \in \Sigma_M} J(K_\eta) = \min_{\eta \in \Sigma_M} \min_{f \in \mathcal{H}_{K_\eta}} \left\{ R(f^n) + \lambda \| \beta \|^2_{\mathcal{H}_{K_\eta}} \right\}$$

- The problem is jointly convex in (η, α) and can be solved efficiently
- The output is both a set of weights η, and a predictor corresponding to the kernel method trained with kernel K_η.
- This method is usually called Multiple Kernel Learning (MKL).

Example: protein annotation

BIOINFORMATICS

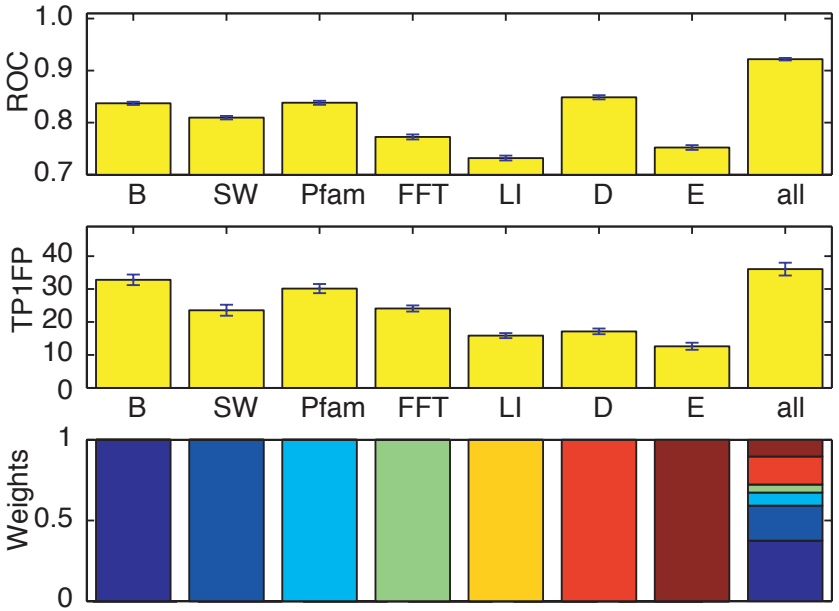
Vol. 20 no. 16 2004, pages 2626–2635
doi:10.1093/bioinformatics/bth294

A statistical framework for genomic data fusion

Gert R. G. Lanckriet[1], Tijl De Bie[3], Nello Cristianini[4], Michael I. Jordan[2] and William Stafford Noble[5,*]

[1]Department of Electrical Engineering and Computer Science, [2]Division of Computer Science, Department of Statistics, University of California, Berkeley 94720, USA, [3]Department of Electrical Engineering, ESAT-SCD, Katholieke Universiteit Leuven 3001, Belgium, [4]Department of Statistics, University of California, Davis 95618, USA and [5]Department of Genome Sciences, University of Washington, Seattle 98195, USA

Kernel	Data	Similarity measure
K_{SW}	protein sequences	Smith-Waterman
K_{B}	protein sequences	BLAST
K_{Pfam}	protein sequences	Pfam HMM
K_{FFT}	hydropathy profile	FFT
K_{LI}	protein interactions	linear kernel
K_{D}	protein interactions	diffusion kernel
K_{E}	gene expression	radial basis kernel
K_{RND}	random numbers	linear kernel

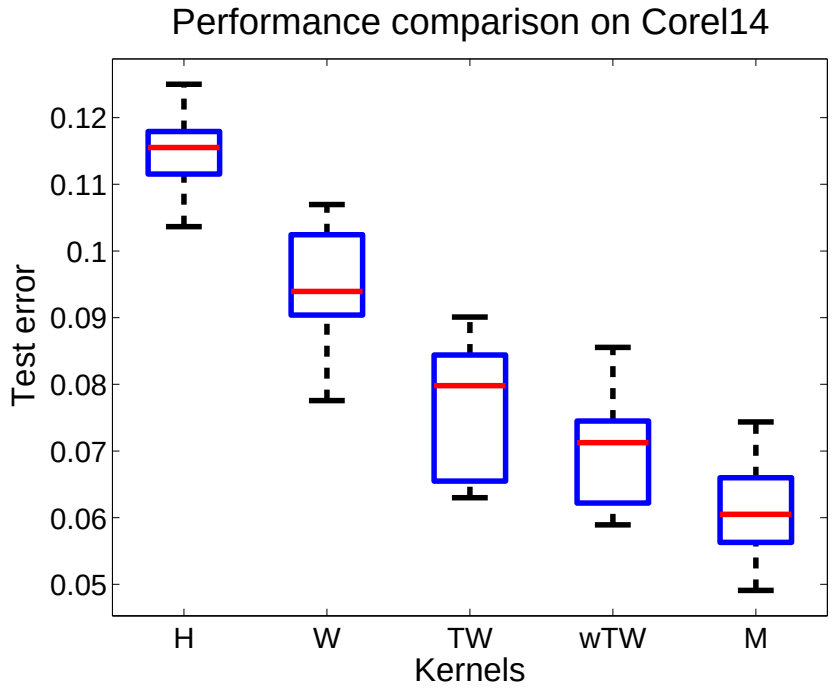


(B) Membrane proteins

Example: 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 by MKL (M).



Sum kernel vs MKL (Bach et al., 2004)

- Learning with the sum kernel (uniform combination) solves

$$\min_{f_1,\ldots,f_M} \left\{ R\left(\sum_{i=1}^{M} f_i^n\right) + \lambda \sum_{i=1}^{M} \| \beta_i \|^2_{\mathcal{H}_{K_i}} \right\} .$$

- Learning with MKL (best convex combination) solves

$$\min_{f_1,\ldots,f_M} \left\{ R\left(\sum_{i=1}^{M} f_i^n\right) + \lambda \left(\sum_{i=1}^{M} \| \beta_i \|_{\mathcal{H}_{K_i}}\right)^2 \right\} .$$

- Although MKL can be thought of as optimizing a convex combination of kernels, it is more correct to think of it as a penalized risk minimization estimator with the group lasso penalty:

$$\Omega(f) = \min_{f_1+\ldots+f_M=f} \sum_{i=1}^{M} \| \beta_i \|_{\mathcal{H}_{K_i}} .$$

Example: ridge vs LASSO regression

- Take $\mathcal{X} = \mathbb{R}^d$, and for $x = (x_1, \ldots, x_d)^\top$ consider the rank-1 kernels:

$$\forall i = 1, \ldots, d\,, \quad K_i\left(x, x'\right) = x_i x_i'\,.$$

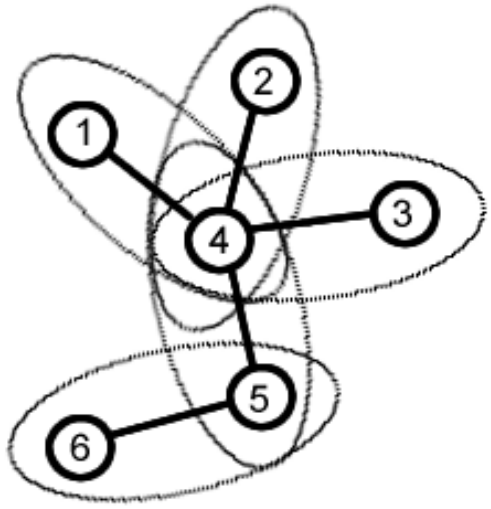
- The sum kernel is $K_S\left(x, x'\right) = \sum_{i=1}^d x_i x_i' = x^\top x$
- Learning with the sum kernel solves a ridge regression problem:

$$\min_{\beta \in \mathbb{R}^d} \left\{ R(X\beta) + \lambda \sum_{i=1}^d \beta_i^2 \right\}\,.$$

- Learning with MKL solves a LASSO regression problem:

$$\min_{\beta \in \mathbb{R}^d} \left\{ R(X\beta) + \lambda \left(\sum_{i=1}^d |\,\beta_i\,| \right)^2 \right\}\,.$$

Example: Graph lasso (Jacob et al., 2009)

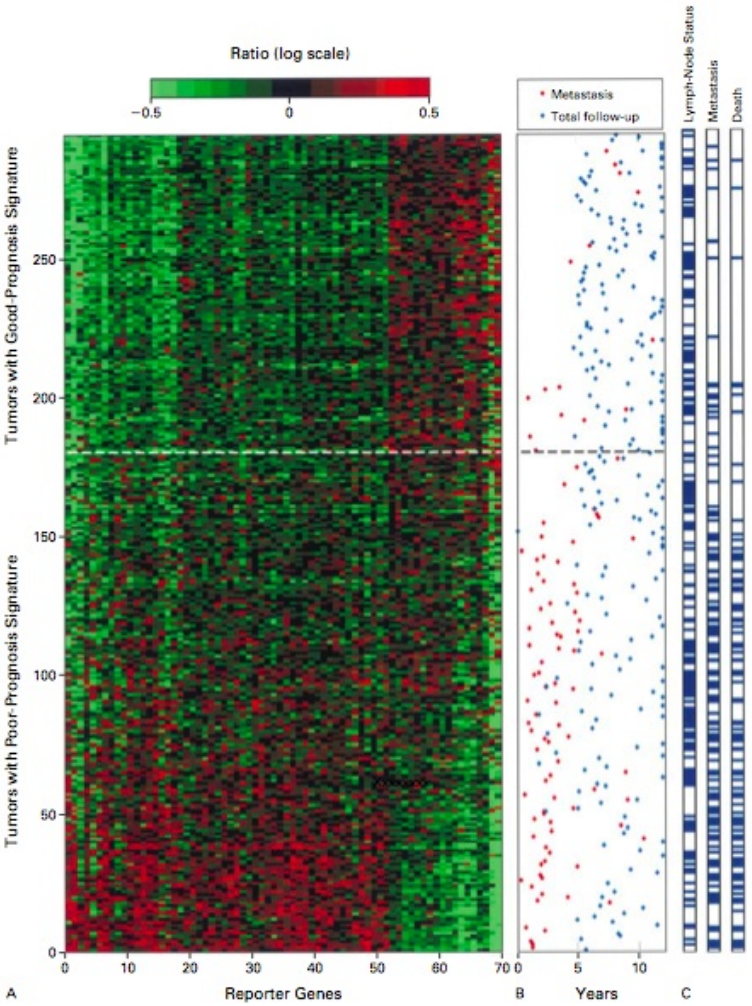


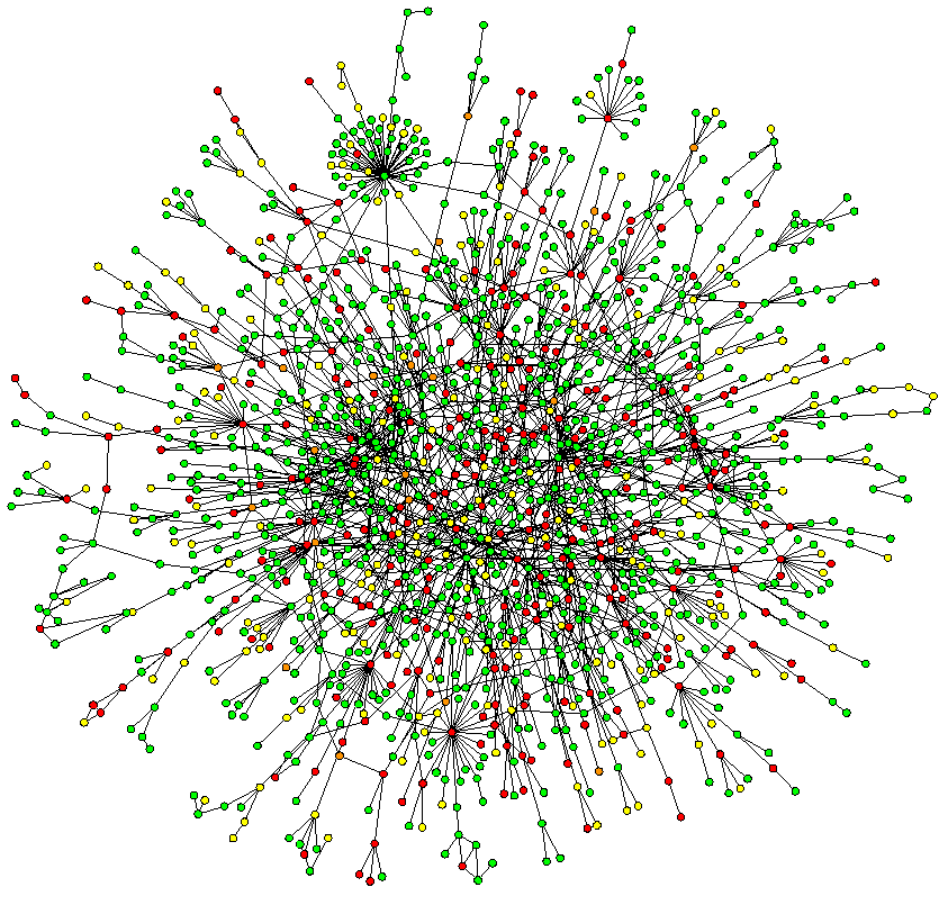
- Graph $G = (V, E)$, $\mathcal{X} = \mathbb{R}^V$
- For each edge $e = (i, j)$, define the kernel

$$K_e(x, x') = x_e^\top x'_e = x_i x'_i + x_j x'_j$$

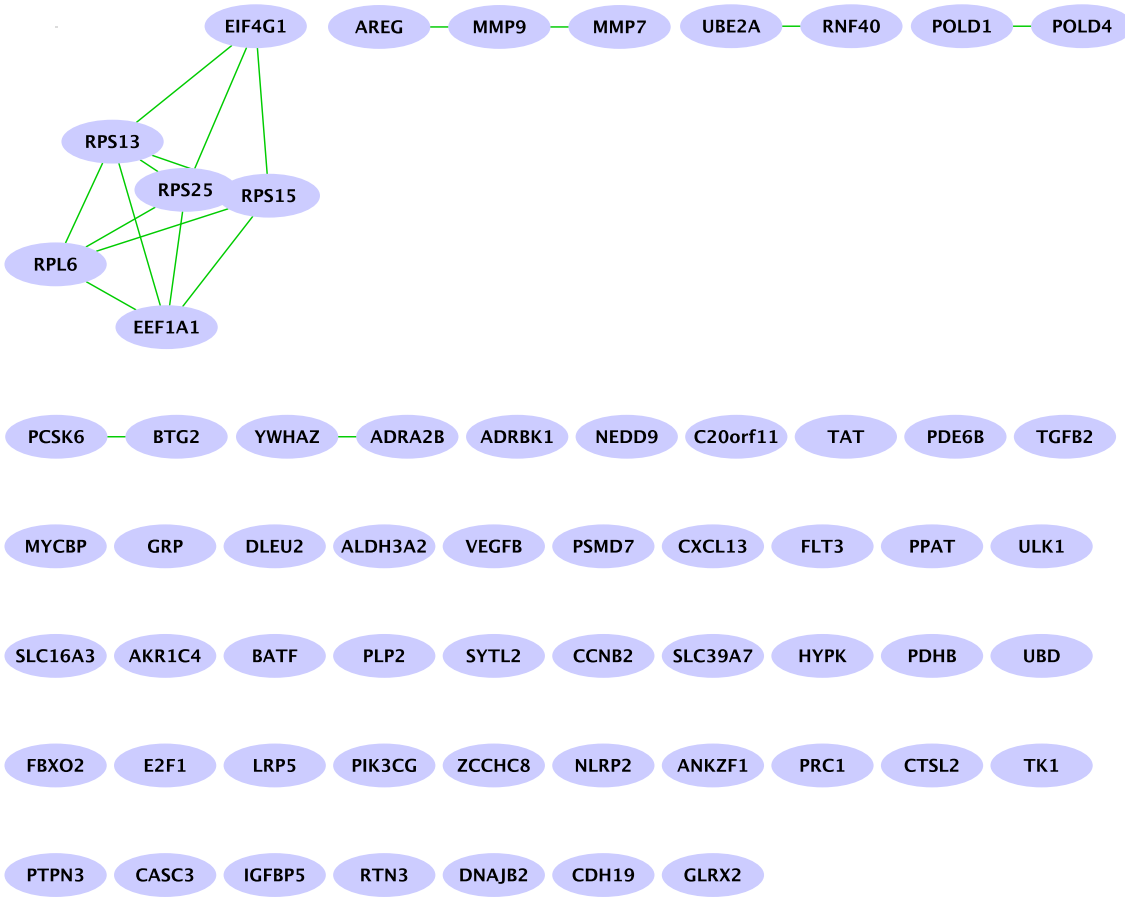
- MKL (aka latent group lasso) with the set $\{K_e : e \in E\}$ leads to a sparse linear model with connected non-zero components.

Application: breast cancer prognosis

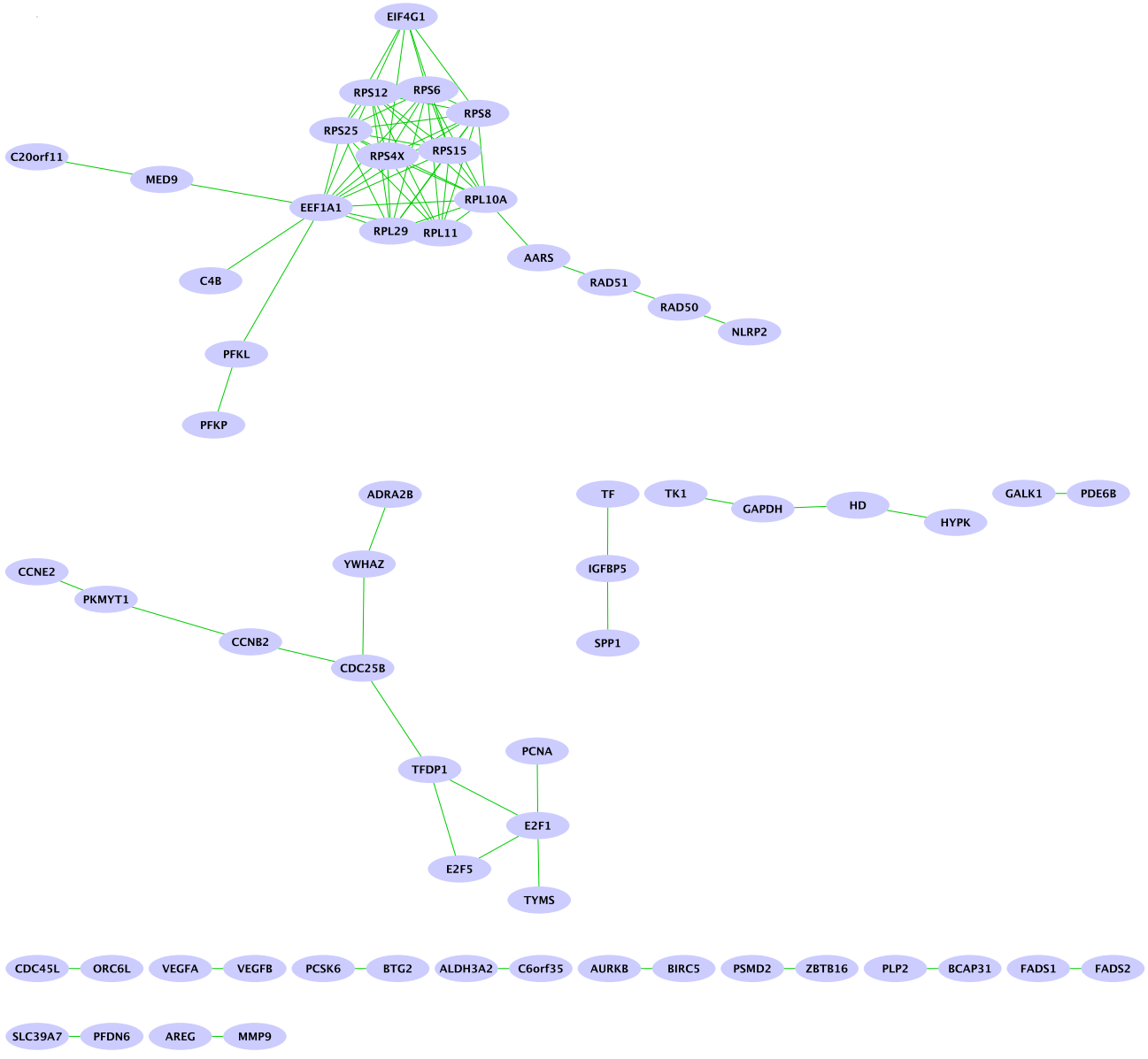




Lasso signature (accuracy 0.61)

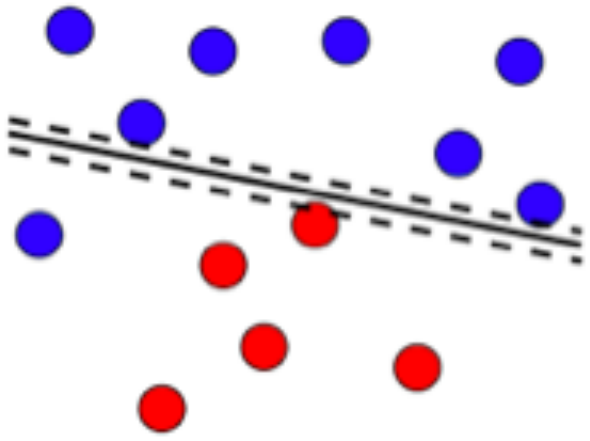


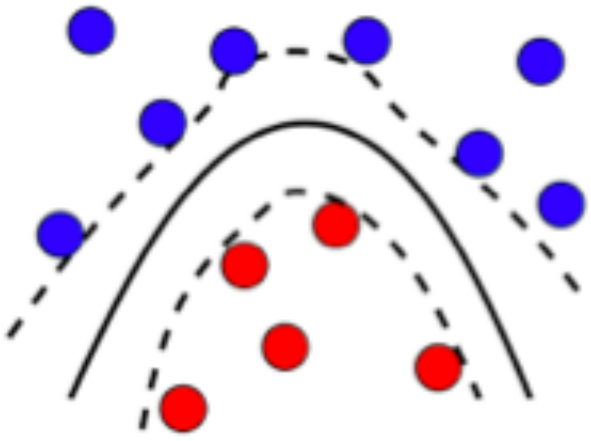
Graph Lasso signature (accuracy 0.64)



Outline

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
- Data integration

References

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

F. R. Bach, G. R. G. Lanckriet, and M. I. Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In *Proceedings of the Twenty-First International Conference on Machine Learning*, page 6, New York, NY, USA, 2004. ACM. doi: http://doi.acm.org/10.1145/1015330.1015424.

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.

References (cont.)

L. Jacob, G. Obozinski, and J.-P. Vert. Group lasso with overlap and graph lasso. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 433–440, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553431. URL http://dx.doi.org/10.1145/1553374.1553431.

G. Lanckriet, N. Cristianini, P. Bartlett, L. El Ghaoui, and M. Jordan. Learning the kernel matrix with semidefinite programming. *J. Mach. Learn. Res.*, 5:27–72, 2004a. URL http://www.jmlr.org/papers/v5/lanckriet04a.html.

G. R. G. Lanckriet, T. De Bie, N. Cristianini, M. I. Jordan, and W. S. Noble. A statistical framework for genomic data fusion. *Bioinformatics*, 20(16):2626–2635, 2004b. doi: 10.1093/bioinformatics/bth294. URL http://bioinformatics.oupjournals.org/cgi/content/abstract/20/16/2626.

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.

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.

References (cont.)

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.

A. Rakotomamonjy, F. Bach, S. Canu, and Y. Grandvalet. SimpleMKL. *J. Mach. Learn. Res.*, 9: 2491–2521, 2008. URL http://jmlr.org/papers/v9/rakotomamonjy08a.html.

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.

F. Rapaport, A. Zynoviev, M. Dutreix, E. Barillot, and J.-P. Vert. Classification of microarray data using gene networks. *BMC Bioinformatics*, 8:35, 2007. doi: 10.1186/1471-2105-8-35. URL http://dx.doi.org/10.1186/1471-2105-8-35.

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.

Y. Yamanishi, J.-P. Vert, and M. Kanehisa. Protein network inference from multiple genomic data: a supervised approach. *Bioinformatics*, 20:i363–i370, 2004. URL http://bioinformatics.oupjournals.org/cgi/reprint/19/suppl_1/i323.