Support Vector Machines and Kernel Methods in bioinformatics

Jean-Philippe Vert
Ecole des Mines de Paris
Computational Biology group
Jean-Philippe.Vert@mines.org

Bioinformatics Center, Kyoto University, Kyoto, Japan, November 18th, 2005.

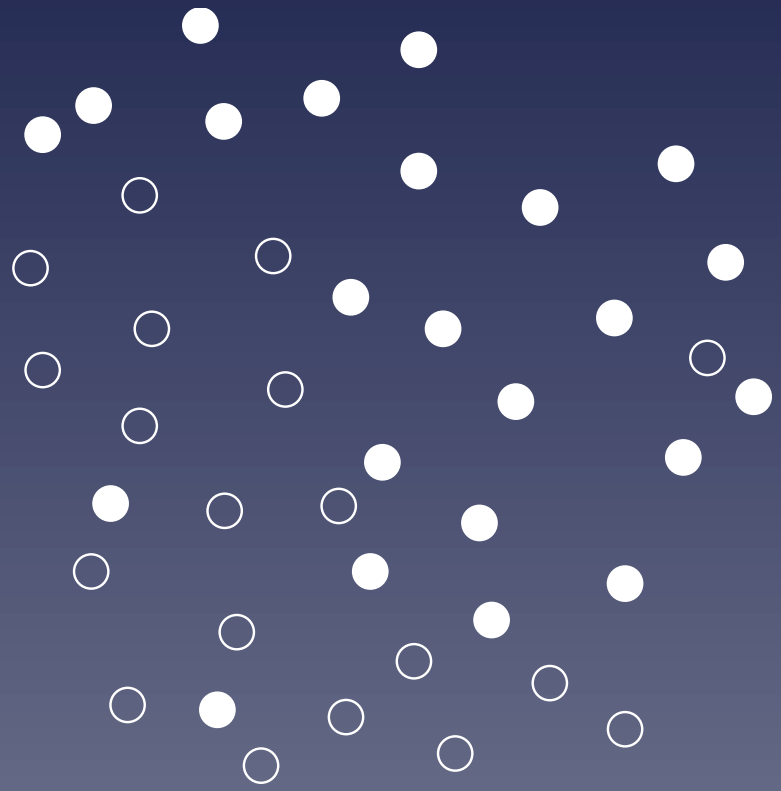
Outline

1. Linear Support Vector Machines (SVM)
2. Non-linear SVMs and kernels
3. Kernels
4. Example: string kernels
5. Example: kernels for TIS
6. Kernel methods

Part 1

Linear Support Vector Machines (SVM)

Pattern recognition



Examples of classification problems

- QSAR and chemoinformatics: x is a molecule, y is a property (active / inactive, toxic / non-toxic...)
- Medical diagnosis: x is a set of features (age, sex, blood type, genome...), y indicates the risk.
- Gene function prediction: x is a string, y is a function

What is a SVM?

- a family of learning algorithm for pattern recognition (works also for more than 2 classes)
- Input: a training set

$$\mathcal{S} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$$

of objects $x_i \in \mathcal{X}$ and their known classes $y_i \in \{-1, +1\}$.

- Output: a classifier $f : \mathcal{X} \to \{-1, +1\}$ which predicts the class $f(x)$ for any (new) object $x \in \mathcal{X}$.

Related approaches

- Bayesian classifier (based on maximum a posteriori probability)
- Fisher linear discriminant
- Neural networks
- Expert systems (rule-based)
- Decision tree
- ...

SVM particularities

- Good performance in real-world applications
- Robust in high dimension (e.g., images, microarray data, texts)
- Handles structured data (sequences, graphs)
- Easy integration of heterogeneous data

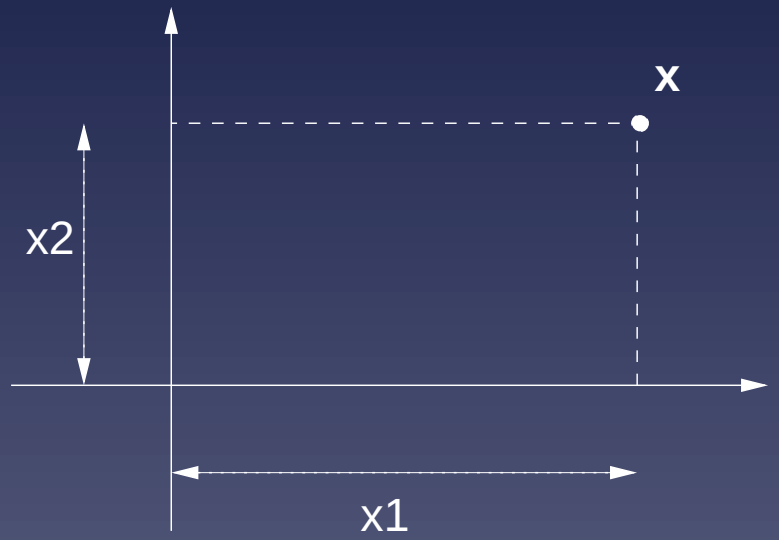
Framework

- We suppose (for now) that the object are finite-dimensional real vectors: $\mathcal{X} = \mathbb{R}^n$ and an object is:

$$\vec{x} = (x_1, \ldots, x_m).$$

- x_i can for example be a feature of a more general object
- Example: a protein sequence can be converted to a 20-dimensional vector by taking the amino-acid composition

Vectors and inner product

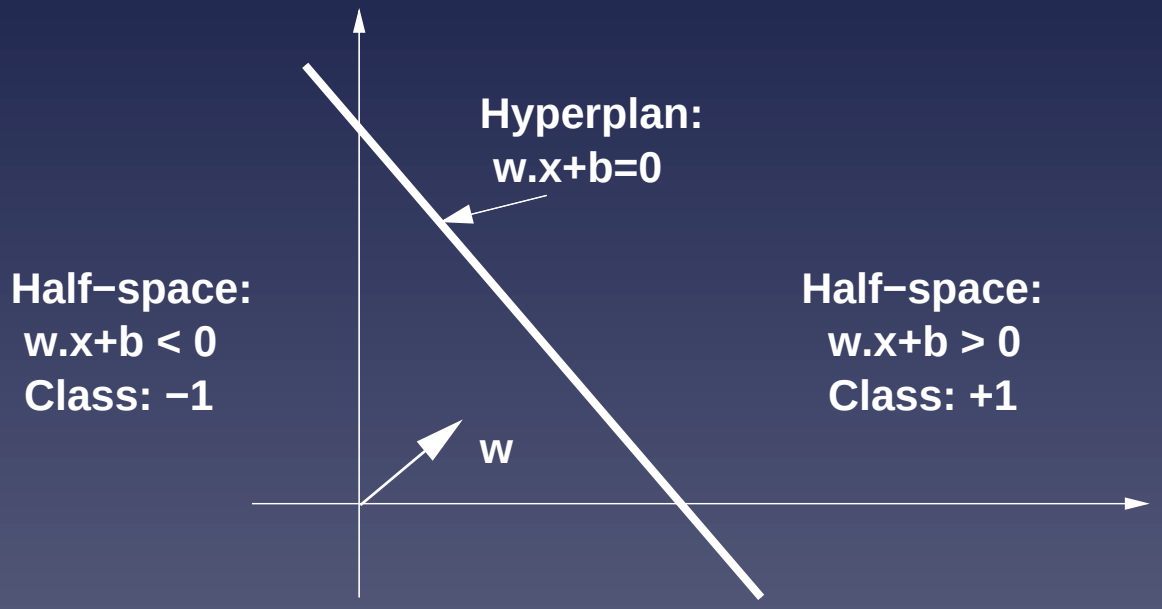


inner product:

$$\vec{x}.\vec{x'} = x_1 x'_1 + x_2 x'_2 \quad (+ \ldots + x_m x'_m) \tag{1}$$

$$= ||\vec{x}||.||\vec{x'}||. \cos(\vec{x}, \vec{x'}) \tag{2}$$

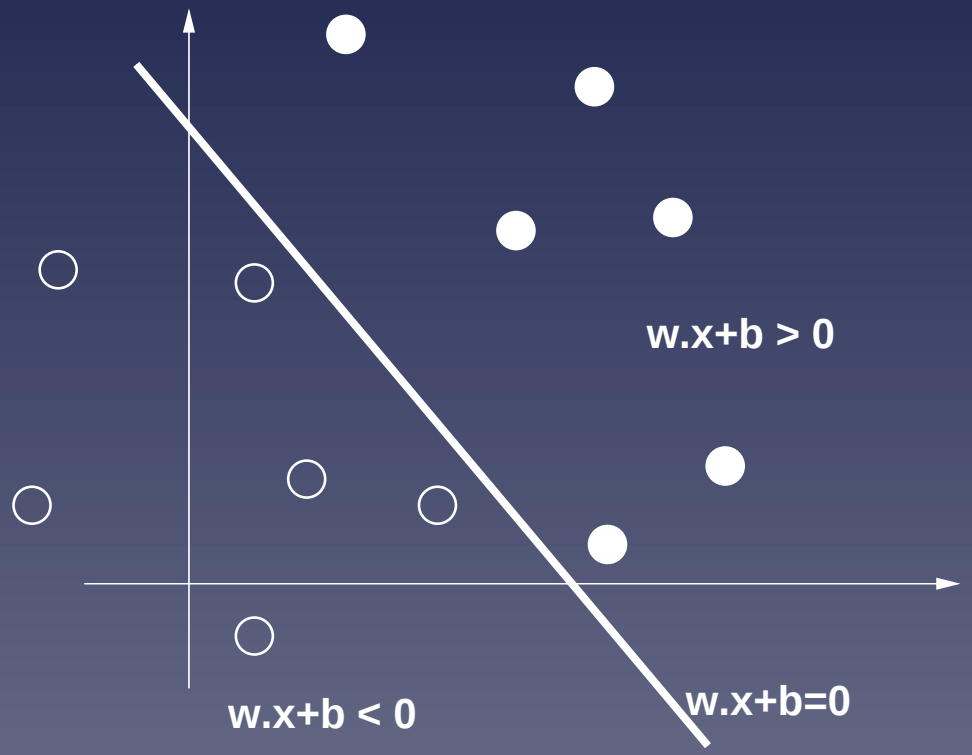
Linear classifier



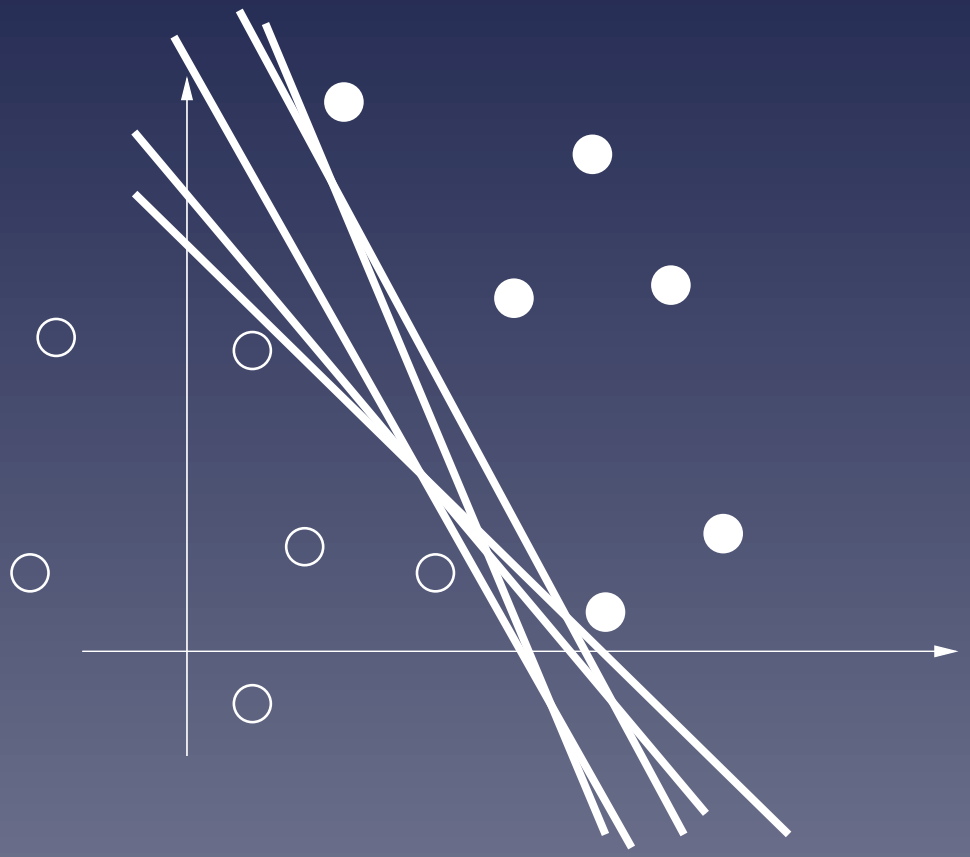
Classification is base on the sign the decision function:

$$f_{\vec{w},b}(\vec{x}) = \vec{w}.\vec{x} + b$$

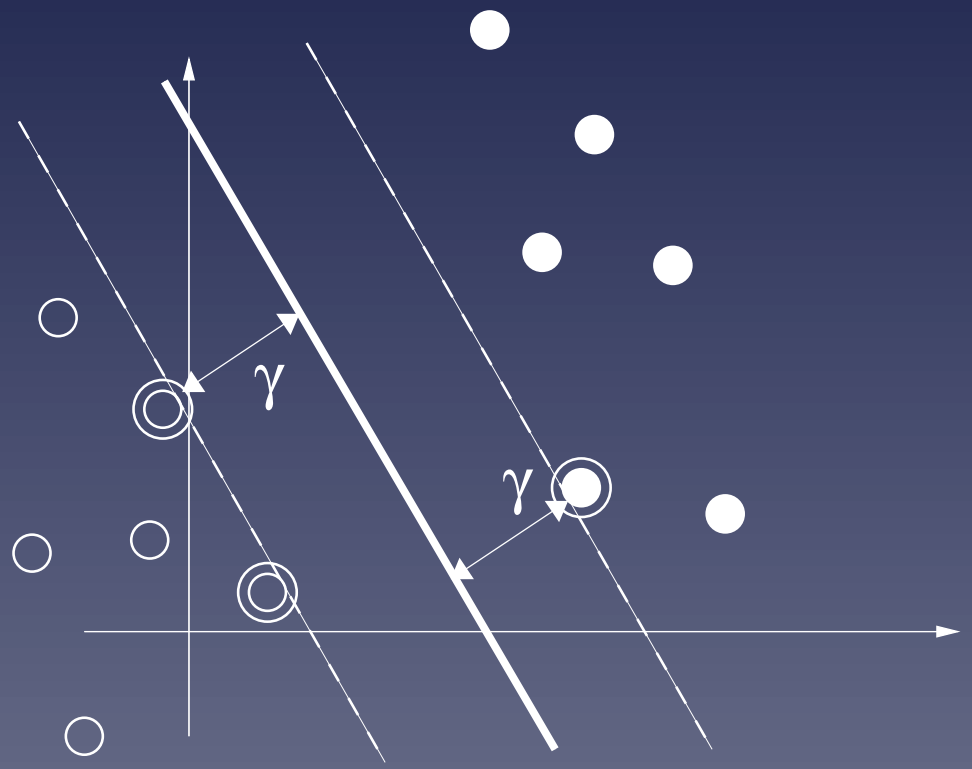
Linearly separable training set



Which one is the best?

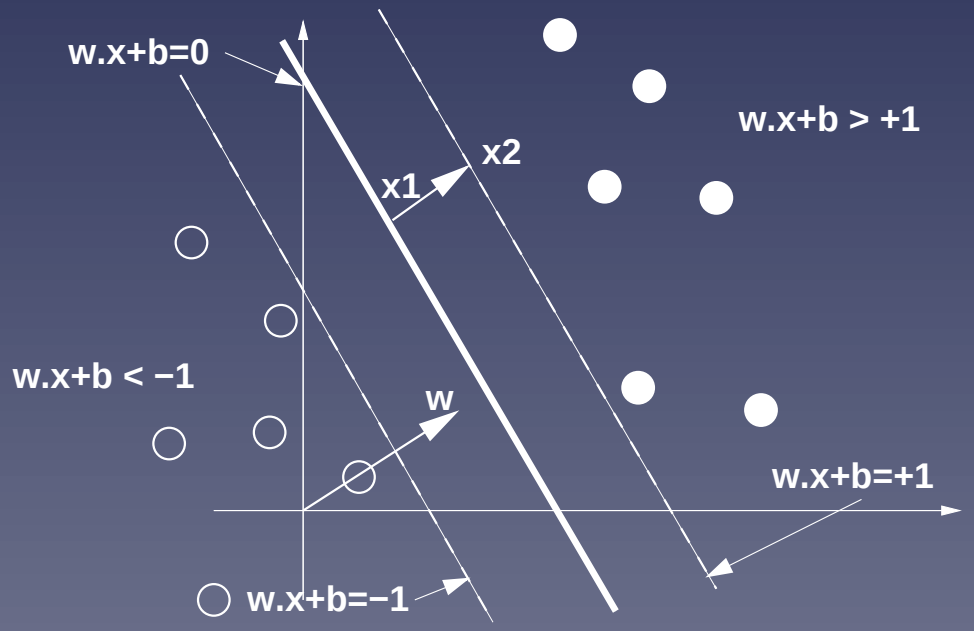


Vapnik's answer : LARGEST MARGIN



How to find the optimal hyperplane?

For a given linear classifier $f_{\vec{w},b}$ consider the tube defined by the values -1 and $+1$ of the decision function:



The width of the tube is $1/||\vec{w}||$

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:

$$\gamma = ||\vec{x}_2 - \vec{x}_1|| = \frac{1}{||\vec{w}||}.$$

All training points should be on the right side of the tube

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 as follows:

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

Finding the optimal hyperplane

The optimal hyperplane is defined by the pair $(\vec{w}, b)$ which solves the following problem:

Minimize:

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

under the constraints:

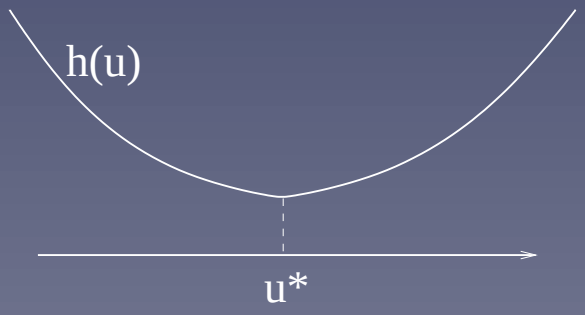
$$\forall i = 1, \ldots, N, \qquad y_i \left(\vec{w}.\vec{x}_i + b\right) - 1 \geq 0.$$

This is a classical quadratic program.

How to find the minimum of a convex function?

If $h(u_1, \ldots, u_n)$ is a convex and differentiable function of n variable, then $\vec{u}^*$ is a minimum if and only if:

$$\nabla h(u^*) = \begin{pmatrix} \frac{\partial h}{\partial u_1}(\vec{u}^*) \\ \vdots \\ \frac{\partial h}{\partial u_1}(\vec{u}^*) \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$$



How to find the minimum of a convex function with linear constraints?

Suppose that we want the minimum of $h(u)$ under the constraints:

$$g_i(\vec{u}) \geq 0, \quad i = 1, \ldots, N,$$

where each function $g_i(\vec{u})$ is affine.

We introduce one variable α_i for each constraint and consider the Lagrangian:

$$L(\vec{u}, \vec{\alpha}) = h(\vec{u}) - \sum_{i=1}^{N} \alpha_i g_i(\vec{u}).$$

Lagrangian method (ctd.)

For each $\vec{\alpha}$ we can look for $\vec{u}_\alpha$ which minimizes $L(\vec{u}, \vec{\alpha})$ (with no constraint), and note the dual function:

$$L(\vec{\alpha}) = \min_{\vec{u}} L(\vec{u}, \vec{\alpha}).$$

The dual variable $\vec{\alpha}^*$ which maximizes $L(\vec{\alpha})$ gives the solution of the primal minimization problem with constraint:

$$\vec{u}^* = \vec{u}_{\alpha^*}.$$

Application to optimal hyperplane

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:

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

Solving the dual problem

The dual problem is to find α^* 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 (simple) constraints $\alpha_i \geq 0$ (for $i = 1, \ldots, N$), and

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

$\vec{\alpha}^*$ can be easily found using classical optimization softwares.

Recovering the optimal hyperplane

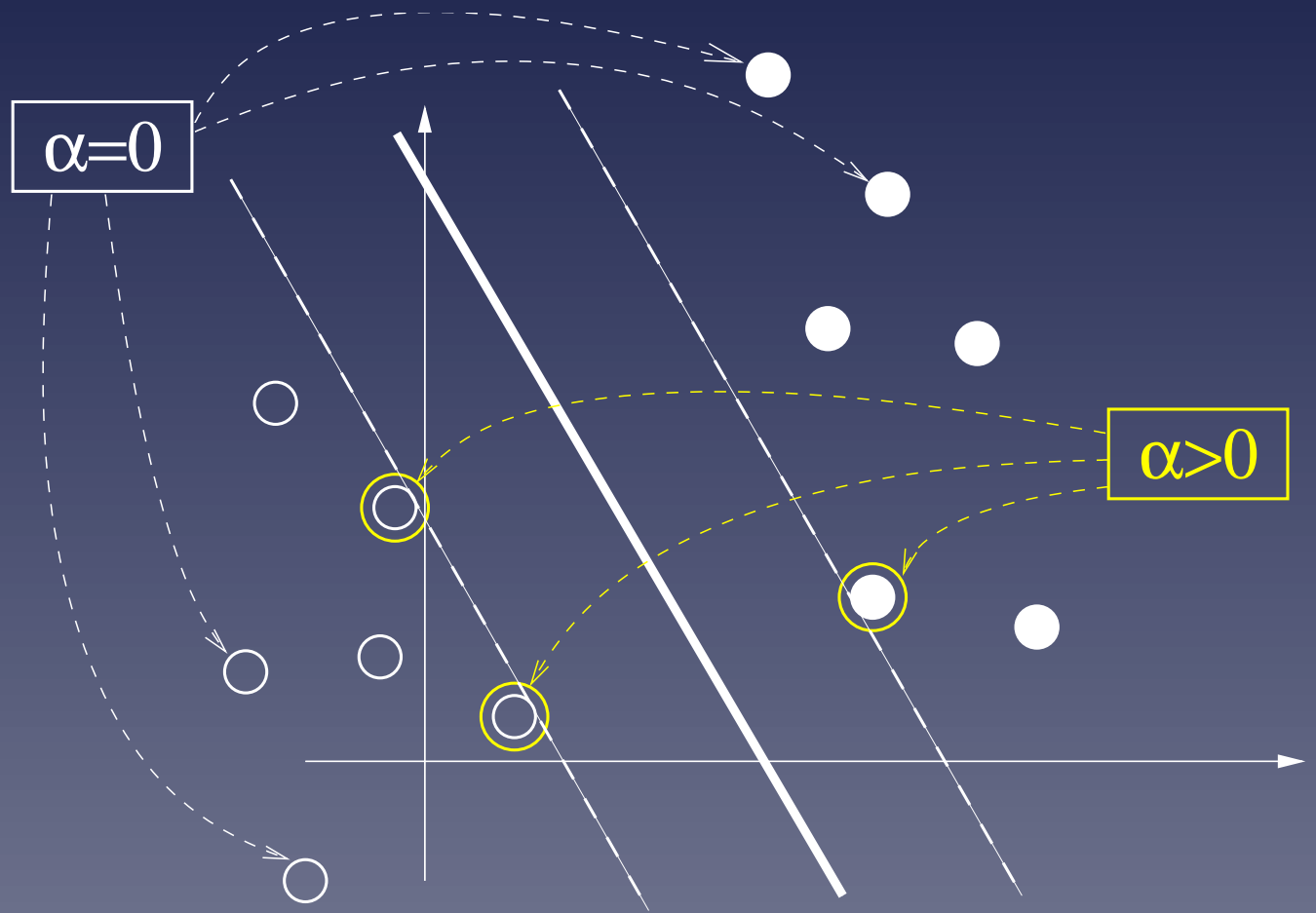
Once $\vec{\alpha}^*$ is found, we recover $(\vec{w}^*, b^*)$ corresponding to the optimal hyperplane. w^* is given by:

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

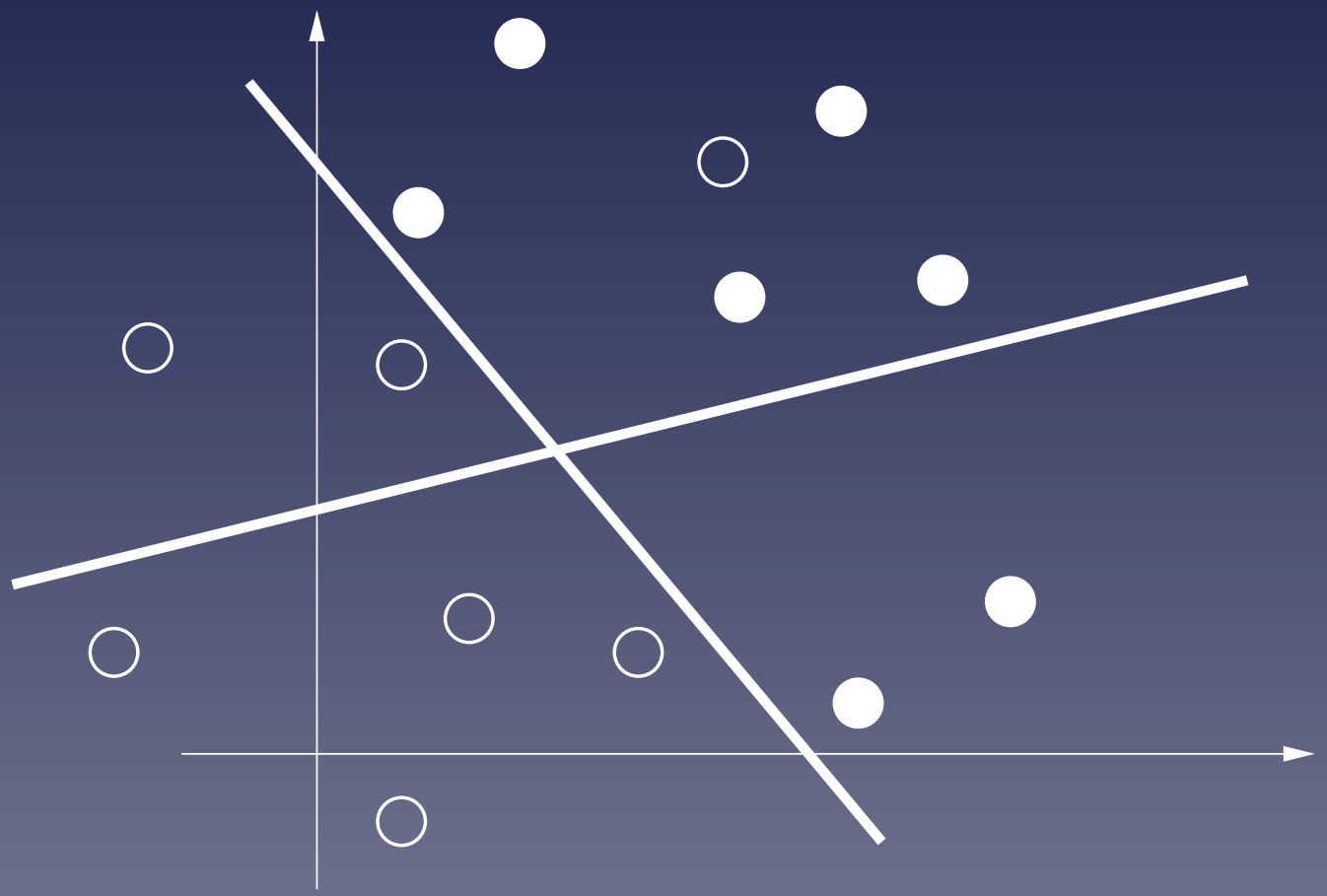
and the decision function is therefore:

$$\begin{aligned} f^*(\vec{x}) &= \vec{w}^*.\vec{x} + b^* \\ &= \sum_{i=1}^{N} \alpha_i \vec{x}_i.\vec{x} + b^*. \end{aligned} \tag{3}$$

Interpretation : support vectors



In general, training sets are not linearly separable



What goes wrong?

The dual problem, 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 $\alpha_i \geq 0$ (for $i = 1, \ldots, N$), and

$$\sum_{i=1}^{N} \alpha_i y_i = 0,$$

has no solution: the larger some α_i, the larger the function to maximize.

Enforcing a solution

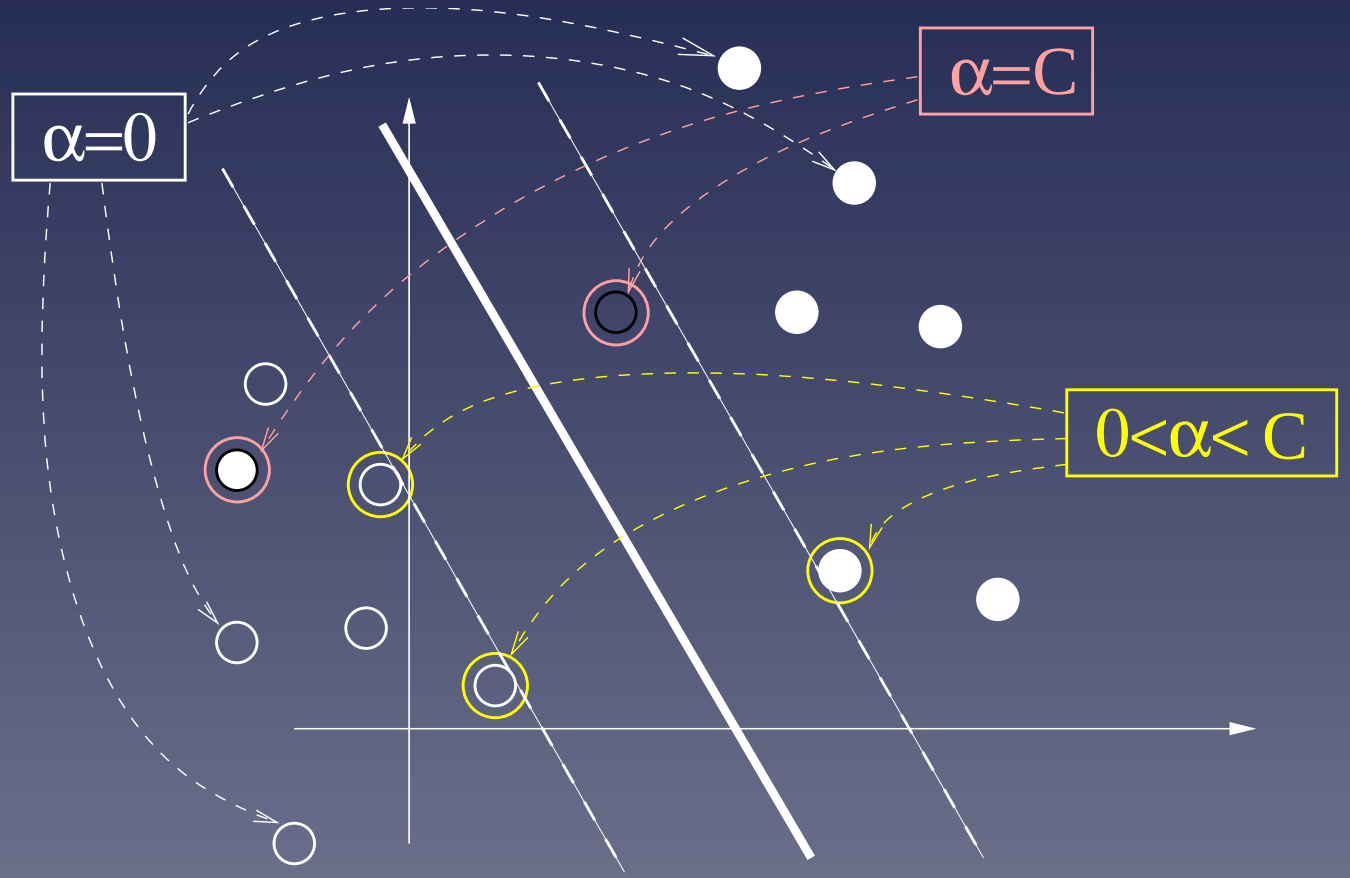
One solution is to limit the range of $\vec{\alpha}$, to be sure that one solution exists. For example, maximize

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

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



Remarks

- This formulation finds a trade-off between:
  - ⋆ minimizing the training error
  - ⋆ maximizing the margin
- Other formulations are possible to adapt SVM to general training sets.
- All properties of the separable case are conserved (support vectors, sparseness, computation efficiency...)

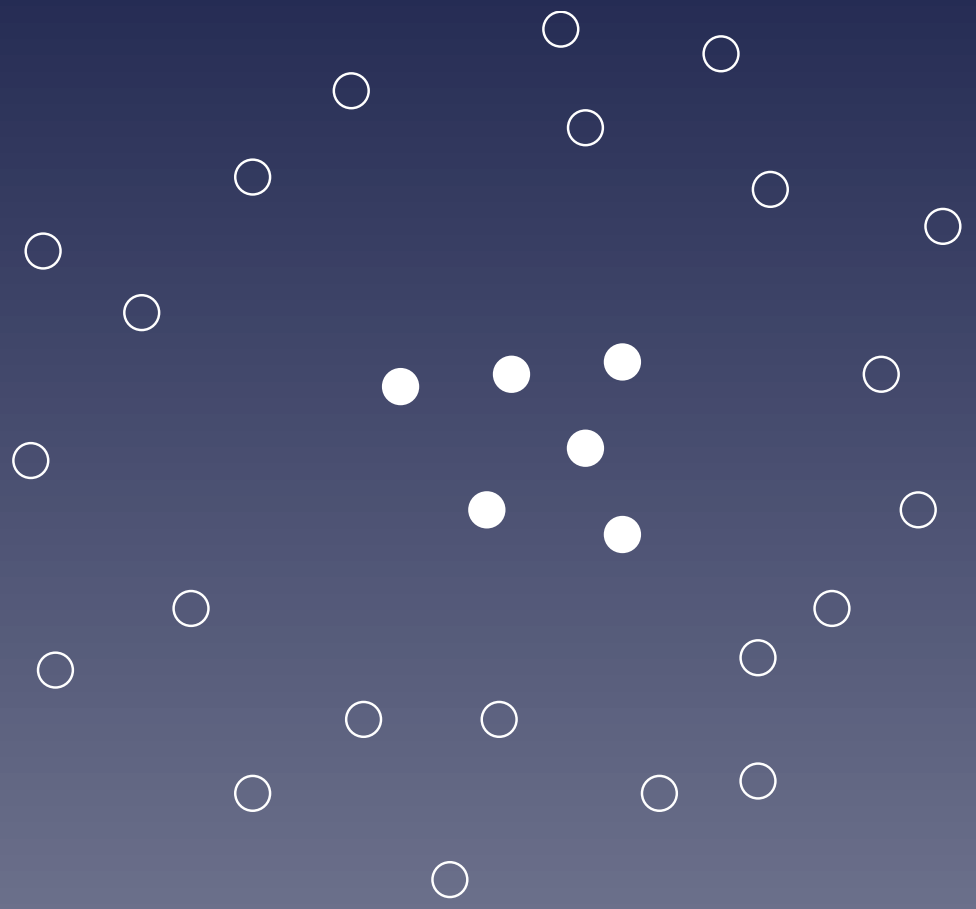
Linear SVM: conclusion

- Finds the optimal hyperplane, which corresponds to the largest margin
- Can be solved easily using a dual formulation
- The solution is sparse: the number of support vectors can be very small compared to the size of the training set
- Only support vectors are important for prediction of future points. All other points can be forgotten.

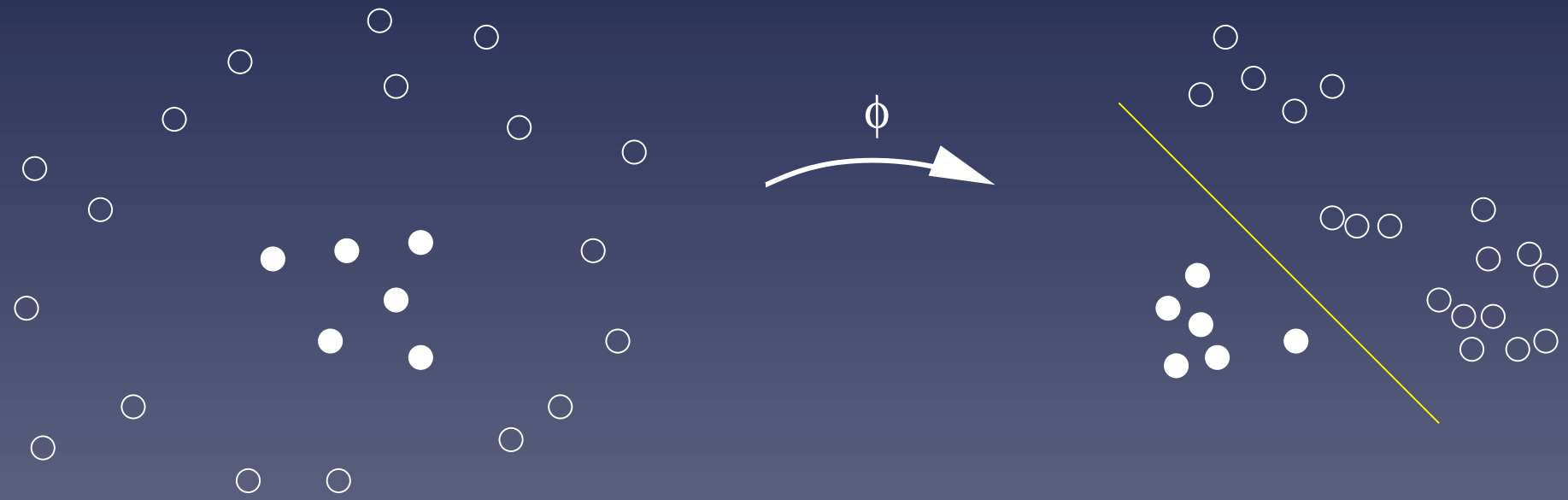
Part 2

Non-linear SVMs and kernels

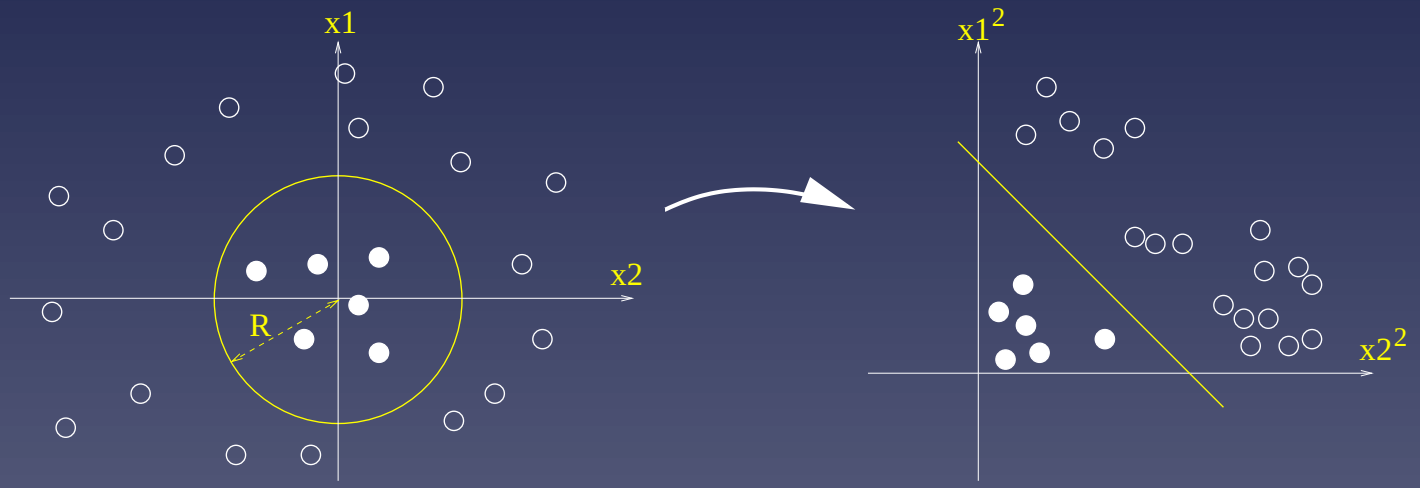
Sometimes linear classifiers are not interesting



Solution: non-linear mapping to a feature space



Example



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

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

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 $K(x, x')$

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:

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

Predicting with a SVM in the feature space

The decision function becomes:

$$\begin{aligned} f(x) &= \vec{w}^*.\vec{\Phi}(x) + b^* \\ &= \sum_{i=1}^{N} \alpha_i K(x_i, x) + b^*. \end{aligned} \tag{4}$$

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

For any vector $\vec{x} = (x_1, x_2)'$, consider the mapping:

$$\Phi(\vec{x}) = \left(x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1\right)'.$$

The associated kernel is:

$$\begin{aligned} K(\vec{x}, \vec{x}') &= \Phi(\vec{x}).\Phi(\vec{x}') \\ &= (x_1x_1' + x_2x_2' + 1)^2 \\ &= (\vec{x}.\vec{x}' + 1)^2 \end{aligned}$$

Classical kernels for vectors

- Polynomial:

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

- Gaussian radial basis function

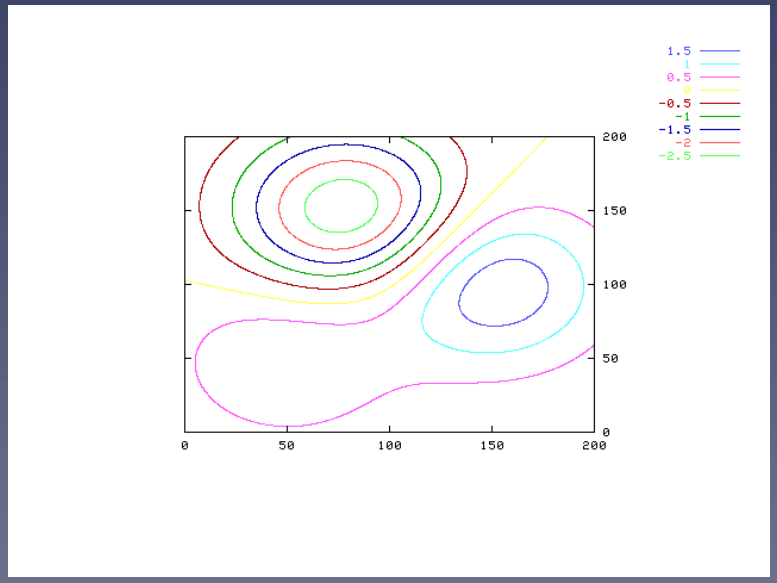
$$K(x, x') = \exp\left(-\frac{||x - x'||^2}{2\sigma^2}\right)$$

- Sigmoid

$$K(x, x') = \tanh(\kappa x.x' + \theta)$$

Example: classification with a Gaussian kernel

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



SVM in practice

- Many free implementations, see http://www.kernel-machines.org

- For example, using GIST (microarray.genomecenter.columbia.edu/gist/)

```
> compute-weights -train data.txt -class data.class > data.weights
> classify -train data.txt -learned data.weights -test test.txt >
test.predict
```

- Parameter tuning is important and not so obvious

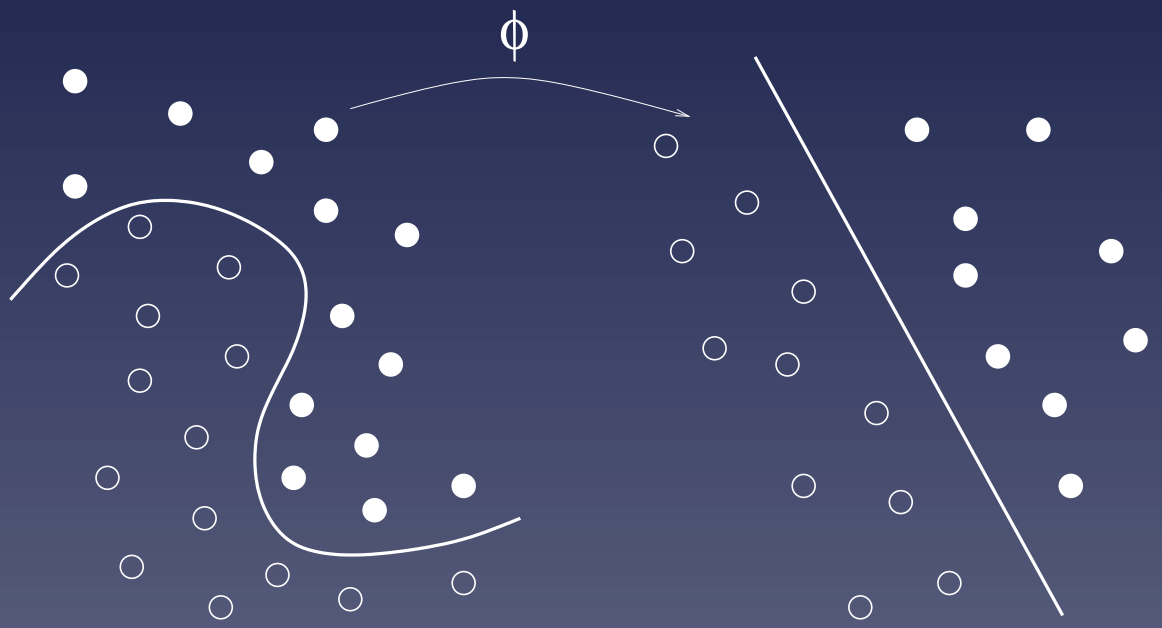
Examples: SVM in bioinformatics

- Gene functional classification from microarry: Brown et al. (2000), Pavlidis et al. (2001)
- Tissue classification from microarray: Mukherje et al. (1999), Furey et al. (2000), Guyon et al. (2001)
- Protein family prediction from sequence: Jaakkoola et al. (1998)
- Protein secondary structure prediction: Hua et al. (2001)
- Protein subcellular localization prediction from sequence: Hua et al. (2001)

Part 3

Kernels

Remember the kernel



$$K(x, x') = \vec{\Phi}(x).\vec{\Phi}(x')$$

Properties of the kernel

- A kernel is a similarity measure
- It defines the geometry of the feature space (lengths and angles)
- (Aronszajn, 1950) A function $K(x, x')$ is a kernel if and only if the following matrix is symmetric positive definite (all eigenvalues are positive) for all choices of $(x_1, \ldots, x_n)$:

$$K = \begin{pmatrix} K(x_1, x_1) & K(x_1, x_2) & \ldots \\ K(x_2, x_1) & K(x_2, x_2) & \ldots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

Important remark

- KERNELS DO NOT NEED TO BE DEFINED FOR VECTORS ONLY.
- KERNELS CAN BE DEFINED FOR STRINGS, GRAPHS, FINITE AUTOMATA, IMAGES, ...
- SVM CAN THEREFORE BE APPLIED AT NO COST ON THESE OBJECTS.

3 ways to make kernels

- Define a set of features of interest, compute the feature vector of every gene, and compute the dot products (see examples in yesterday's talk).
- Define a large set of features and find tricks to compute the dot product implicitly (without computing the feature vectors)
- Start with a similarity measure you find pertinent (e.g., SW score) and check that it is a kernel.

Kernel engineering

Particular kernels can be imagined to include prior knowledge about:

- the types of data (vectors, sequences, graphs...)
- the problem at hand

into the geometry of the feature space.

This process is called kernel engineering

Examples of kernel engineering

- Kernels for sequences based on common subsequences
- Kernel to recognize translation initiation site
- Convolution kernels
- Kernels built from Bayesian tree models
- Diffusion kernels on graphs

Data integration (IMPORTANT)

- Suppose various data (gene sequence, expression, phylogenetic profile...) can be represented by kernels $K_1, \ldots, K_p$.
- Many operations can create new kernels from kernels: sum, pointwise limit,...
- Example: $K = \sum_{i=1}^{p} a_i K_i$ with $a_i \geq 0$ is a new kernel
- The weights a_i can be optimized (semi-definite programming...)

The kernel phylosophy

- Let $\mathcal{F}$ the set of symetric positive definite matrices (or functions)
- Each dataset is represented by a point in $\mathcal{F}$
- The data are then forgotten : everything takes place in $\mathcal{F}$
- $\mathcal{F}$ is a closed convex cone, closed under pointwise limits and Schur products...

Part 4

Example: string kernels based on common subsequences

Motivation

- Goal: define a kernel for variable-length sequences (useful to handle bio-polymers)
- Intuition: two sequences are related when they share common substrings or subsequences.

References

- H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini and C. Watkins. **Text classification using string kernels**. *Journal of Machine Learning Research*, 2:419-444, 2002.

- C. Leslie, E. Eskin and W.S. Noble. **The spectrum kernel: a string kernel for svm protein classification**. Russ B. Altman, A. Keith Dunker, Lawrence Hunter, Kevin Lauerdale, Teri E. Klein, , *Proceedings of the Pacific Symposium on Biocomputing 2002*, 564-575. World Scientific, 2002.

Substrings

- A string $s = s_1, \ldots, s_p$ is a substring of a string $x = x_1, \ldots, x_n$ (with $n \geq p$) if the letters of s appear in the same order in x (gaps allowed).
- The length $l(s, x)$ of a substring s in a string x is the distance between the first and the last letter in x
- Example: $s =$ `ofot` is a substring of $x =$ `bioinformatics`, with length $l(s, x) = 9$.

String matching kernel (Lohdi et al., 2002)

- The string matching kernel is defined by:

$$K(x, x') = \sum_{s \text{ common substring}} \lambda^{l(s,x)+l(s,x')},$$

  where λ is a parameter.

- Two strings are similar when they share many common substrings

- The feature space is the space of all possible substrings

Computation of the string matching kernel

- The dimension of the feature space is very large (number of possible substrings), but...
- There exists a dynamic programming method to compute the kernel $K(x, x')$ between any two sequences in $O(|x||x'|n)$, where n is the length of the substrings considered.
- Promising results on text classification

Spectrum kernel (Leslie et al., 2002)

- Same idea, but gaps not allowed (common sub-blocks)
- Efficient implementation using a suffix tree
- Classification of a sequence x in $O(|x|)$ using a sliding window
- Encouraging results on remote homology detection (superfamily prediction): performs like PSI-Blast, a bit lower than SAM and SVM+Fisher kernel

More string kernels

- Mismatch kernel
- Fisher kernel
- Convolution and local alignment kernels
- Motif kernel

Part 5

Kernel to recognize translation initiation site

The problem

- Translation initiation sites (TIS) are the position in DNA where regions coding for proteins start
- All coding sequences start with the start codon `ATG`
- Given a `ATG` in a DNA sequence, is it a TIS?

References

- A. Zien, G. Ratsch, S. Mika, B. Schölkopf, T. Lengauer and K.-R. Muller. **Engineering support vector machine kernels that recognize translation initiation sites**. *Bioinformatics*, 16(9):799-807, 2000.

Formulation

- Pick up a window of 200 nucleotides centered around the candidate ATG
- Encode each nucleotide with a 5 bits word: 00001,. . . ,10000 for A,C,G,T and unknown.
- Use this 1000 long bit vectors to train a SVM to predict whether the central ATG corresponds to a TIS
- Which kernel to use?

Polynomial kernels

$$K(\vec{x}, \vec{x}') = (\vec{x}.\vec{x}')^d$$

The corresponding feature space is made of C_{n-1}^d monomials features of degree d

- $d = 1$: counts the number of common bits
- $d = 2$: counts the number of common pairs of bits (pairwise correlations)
- etc...

Locally improved kernels

- Intuition: while certain local correlations are typical for TIS, dependencies between distant positions are of minor importance or do not even exist. They only add noise to the feature space.
- At each sequence position, sequences can be compared locally using a small window of length $2l+1$ with inner correlations of up to d_1 positions:

$$win_p(x,x') = \left(\sum_{j=-l}^{+l} w_j \mathsf{match}_{p+j}(x,x') \right)^{d_1} .$$

Locally improved kernels (ctd.)

- Add the contributions of all windows, and of correlations between up to d_2 windows:

$$K(x, x') = \left(\sum_{p=1}^{n} win_p(x, y) \right)^{d_2}$$

Results

$d_2 > 1$ (long-range correlations) does not improve performance

Method	Overall error (%)
Neural network	15.4
Salzberg method	13.8
SVM, linear kernel	13.2
SVM, locally improved kernel ($d_1 = 4$, $l = 4$)	11.9

Part 6

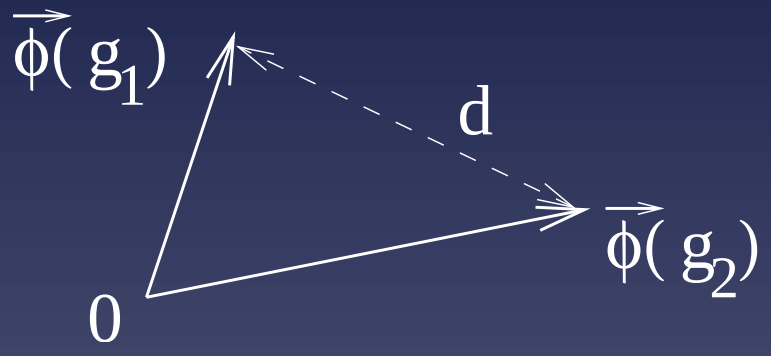
Kernel methods

Kernel methods

Suppose you are given a kernel $K(.,.)$. Then you can perform various operations in the feature space without computing the image $\vec{\Phi}(g)$ of each gene g:

- Compute the distance between any two genes, or between any gene and the center of mass of the gene database
- Principal component analysis (PCA)
- Canonical correlation analysis (CCA)
- Classify the genes into classes (Support vector machines)

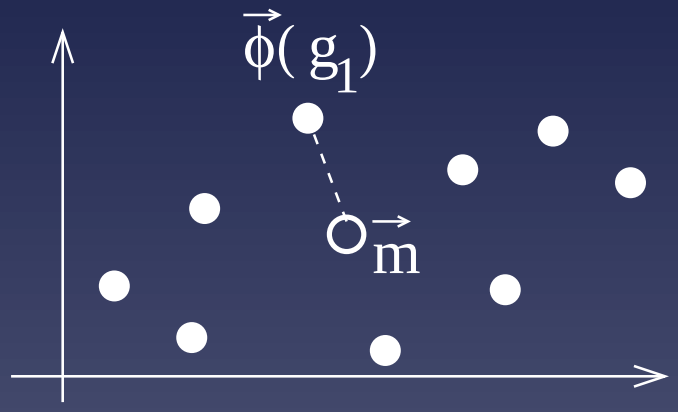
Distance between two genes



$$
\begin{aligned}
d(g_1, g_2)^2 &= \|\vec{\Phi}(g_1) - \vec{\Phi}(g_2)\|^2 \\
&= \left(\vec{\Phi}(g_1) - \vec{\Phi}(g_2)\right) . \left(\vec{\Phi}(g_1) - \vec{\Phi}(g_2)\right) \\
&= \vec{\Phi}(g_1).\vec{\Phi}(g_1) + \vec{\Phi}(g_2).\vec{\Phi}(g_2) - 2\vec{\Phi}(g_1).\vec{\Phi}(g_2)
\end{aligned}
$$

$$
d(g_1, g_2)^2 = K(g_1, g_1) + K(g_2, g_2) - 2K(g_1, g_2)
$$

Distance between a gene and the center of mass



Center of mass: $\vec{m} = \frac{1}{N}\sum_{i=1}^{N} \vec{\Phi}(g_i)$, hence:

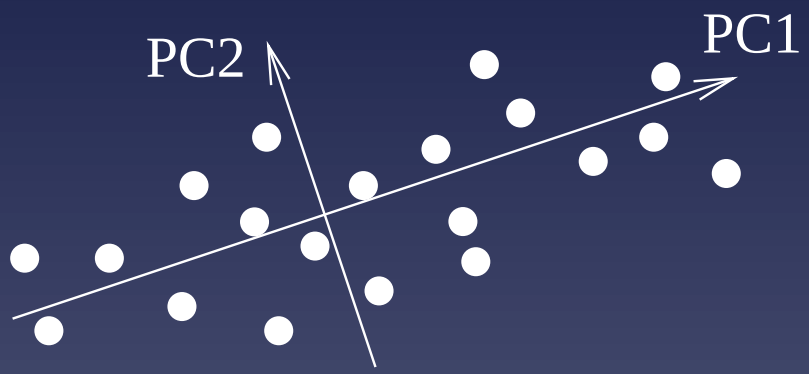
$$\|\vec{\Phi}(g_1) - \vec{m}\|^2 = \vec{\Phi}(g_1).\vec{\Phi}(g_1) - 2\vec{\Phi}(g_1).\vec{m} + \vec{m}.\vec{m}$$

$$= K(g_1, g_1) - \frac{2}{N}\sum_{i=1}^{N} K(g_1, g_i) + \frac{1}{N^2}\sum_{i,j=1}^{N} K(g_i, g_j)$$

Example: greedy multiple alignment (Gorodkin et al., GIW 2001)

- Use the SW score as a kernel for sequences (?)
- Compute the distance between each sequence and the center of mass
- First align the sequences near the center of mass
- Then add sequences one by one to the multiple alignment, by increasing distance from the center of mass

Principal component analysis (PCA)

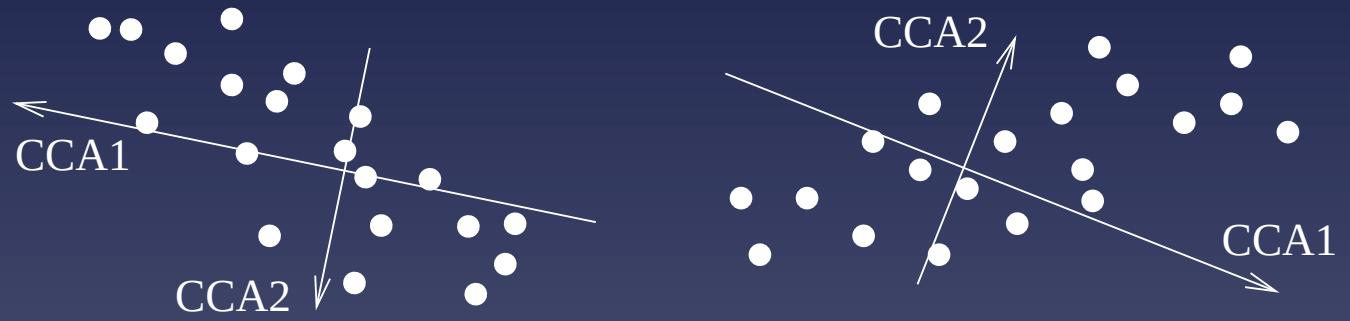


Find the eigenvectors of the matrix:

$$K = \left(\vec{\Phi}(g_i).\vec{\Phi}(g_j)\right)_{i,j=1\ldots N}$$
$$= \left(K(g_i, g_j)\right)_{i,j=1\ldots N}$$

Useful to represent the objects as small vectors (feature extraction).

Canonical correlation analysis (CCA)



K_1 and K_2 are two different kernels for the same objects (genes).
CCA is performed by solving the generalized eigenvalue problem:

$$\begin{pmatrix} 0 & K_1K_2 \\ K_2K_1 & 0 \end{pmatrix} \vec{\xi} = \rho \begin{pmatrix} K_1^2 & 0 \\ 0 & K_2^2 \end{pmatrix} \vec{\xi}$$

Useful to find correlations between different representations of the

same objects

More kernel methods

- Any algorithm can be kernelized if it can be expressed in terms of inner product
- The library of kernel methods include SVM, kernel-PCA, kernel-CCA, kernel-Fisher discriminant, kernel-ICA, kernel-clustering, kernel logistic regression, kernel network inference...
- Modularity : any kernel can be used with any kernel method

Conclusion

Conclusion

- SVM and kernel methods are now widely used in computational biology
- Good performance, possibility to handle and integrate structured data
- Active research field

References

- Schölkopf, B., Tsuda, K., and Vert, J.-P. (2004). Kernel Methods in Computational Biology. *MIT Press*.

- 350+ references listed at:

  `http://cg.ensmp.fr/~vert/svn/bibli/html/biosvm.html`