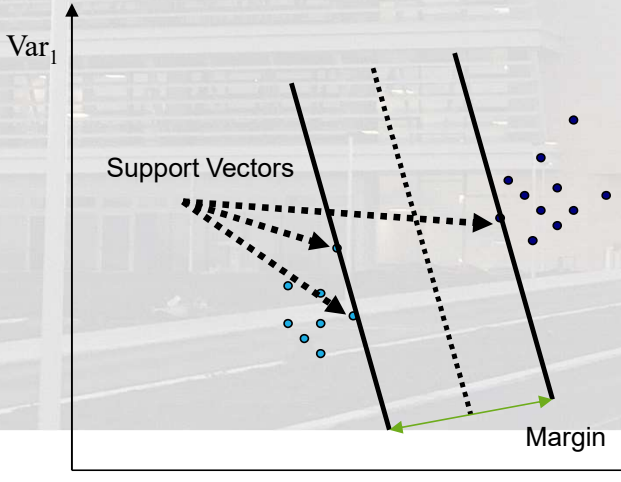
KERNEL MACHINES AND KNOWLEDGE REPRESENTATION

INTRODUCTION

SUPPORT VECTOR MACHINES

- Support Vector Machines (SVMs) are a machine learning paradigm based on the statistical learning theory [Vapnik, 1995]
  - No need to remember everything, just the discriminating instances (i.e. the support vectors, SV)
  - The classifier corresponds to the linear combination of SVs



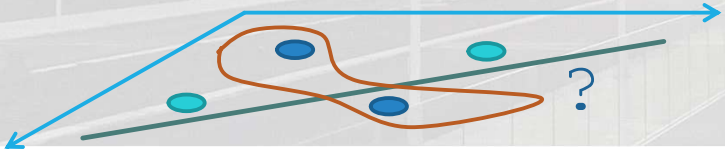
$$h(x) = sgn(\vec{w} \cdot \vec{x} + b) = sgn(\sum_{J=1}^{l} \alpha_j y_j \vec{x_j} \cdot \vec{x} + b)$$

Support Vectors

Only the dot product is required

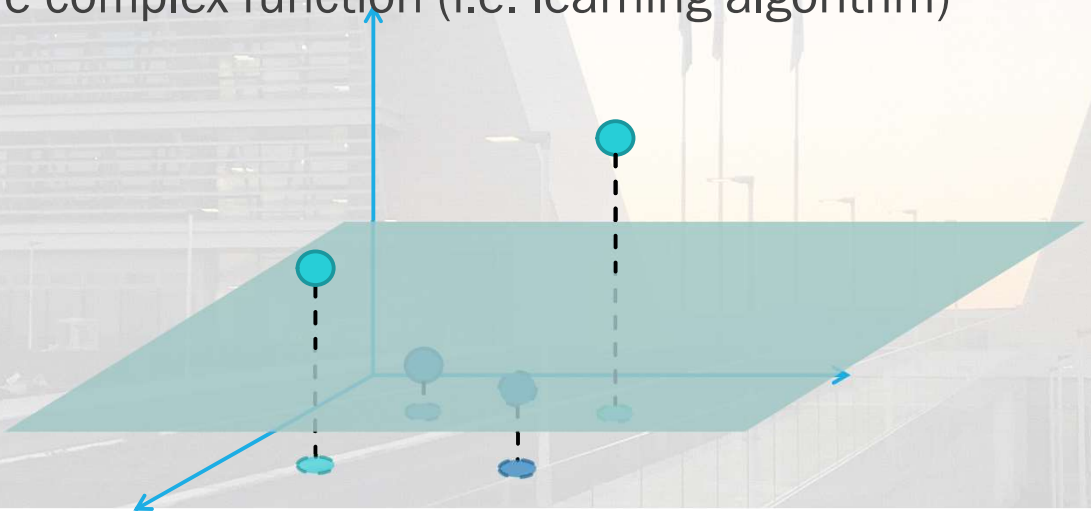
LINEAR CLASSIFIERS AND SEPARABILITY

- In a R^2 space, 3 point can always be separable by a linear classifier
  - but 4 points cannot always be shattered [Vapnik and Chervonenkis(1971)]
- One solution could be a more complex classifier
  - ☹ Risk of over-fitting



LINEAR CLASSIFIERS AND SEPARABILITY (2)

- … but things change when projecting instances in a higher dimension feature space through a function ϕ
- IDEA: It is better to have a more complex feature space instead a more complex function (i.e. learning algorithm)



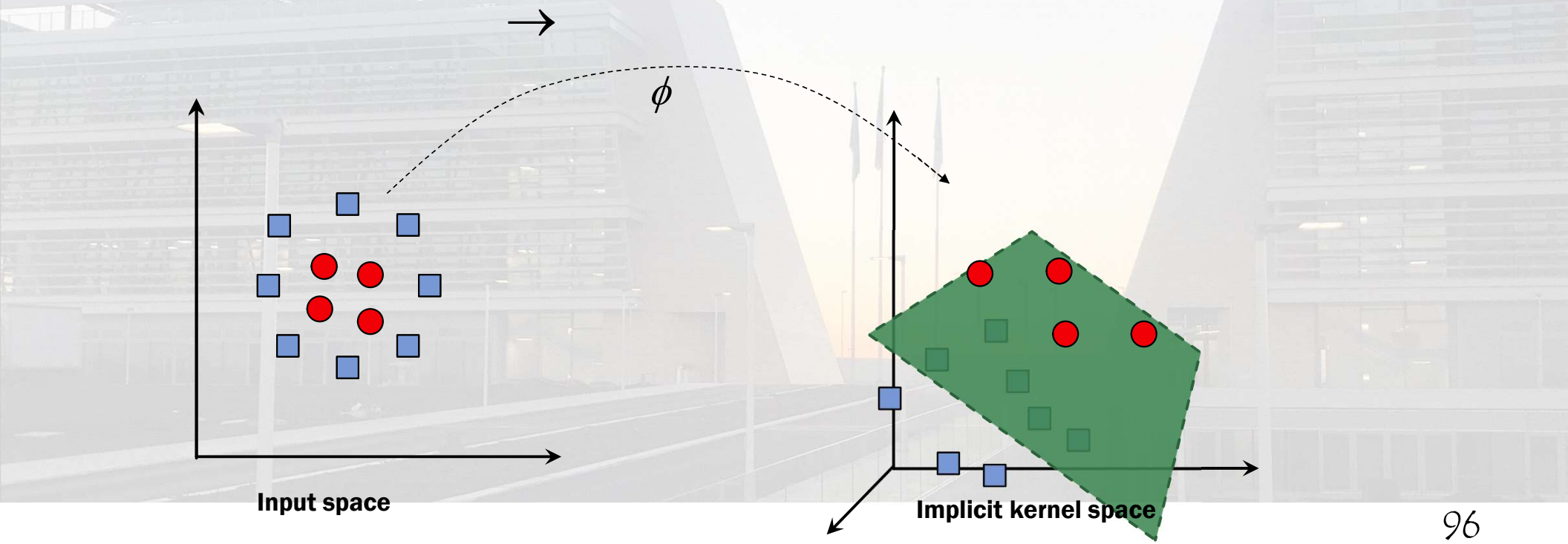
THE KERNEL FUNCTION

- In perceptrons and SVMs the learning algorithm only depends on the scalar product over pairs of example instance vectors
- Basically only the Gram-matrix is involved. In general, we call kernel the following function:

$$K(\vec{x}, \vec{z}) = \Phi(\vec{x}) \cdot \Phi(\vec{z})$$

- The kernel corresponds to a scalar product over the transformed of initial objects x and z
- If the mapping ϕ corresponds to the identity then the kernel is equal to the standard scalar product.
- Notice that the training in most learning machines (such as the perceptron) makes use of instances only through the kernel

FIRST ADVANTAGE: MAKING INSTANCES LINEARLY SEPARABLE



AN EXAMPLE: A MAPPING FUNCTION

- Two masses m_1 and m_2 , one is constrained
- A force f_a is applied to the mass m_1
- Instead of applying an analyitical law we want to experiment
  - The Features of individual experiments are masses m_1, m_2 and the appropriate orce f_a
- It is clear that the Newton law of gravity is involved:

$$f(m_1, m_2, r) = C\frac{m_1 m_2}{r^2}$$

- The task corresponds to determine if $f(m_1, m_2, r) < f_a$

AN EXAMPLE: A MAPPING FUNCTION (2)

$$\vec{x} = (x_1, \dots, x_n) \rightarrow \Phi(\vec{x}) = (\Phi_1(\vec{x}), \dots, \Phi_k(\vec{x}))$$

- This law cannot be expressed linearly. A change of space:

$$(f_a, m_1, m_2, r) \rightarrow (k, x, y, z) = (\ln f_a, \ln m_1, \ln m_2, \ln r)$$

- holds as:

$$\ln f\,(m_1, m_2, r) = \ln C + \ln m_1 + \ln m_2 - 2 \ln r = c + x + y - 2z$$

- The following hyperplane is the requested function *h()*:

$$\ln f_a - \ln m_1 - \ln m_2 + 2 \ln r - \ln C = 0$$

(1,1,-2,-1)·(ln m_1,ln m_2,ln r, ln f_a)+ ln C = 0,

We can decide with no error if masses m_1, m_2 get closer or not

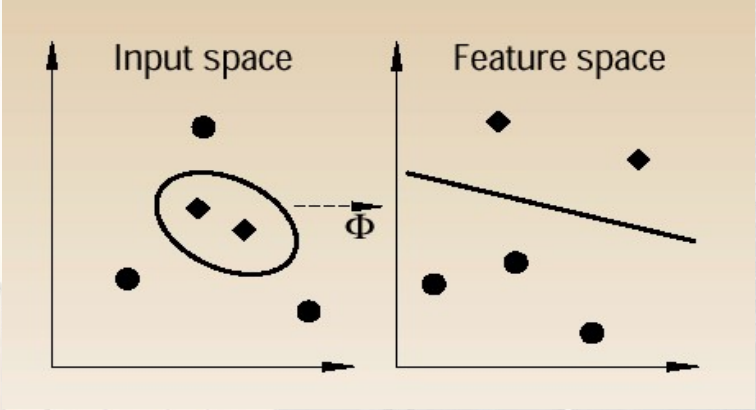
FEATURE SPACES AND KERNELS

- Feature Space
  - The input space is mapped into a new space *F* with scalar product (called *feature space*) through a (non linear) trasformation ϕ

- The kernel function $\phi = R^N \rightarrow F$
  - The evaluation require the computation of the scalar product over the trasformed vectors $\phi(x)$ but not the feature vectors themselves

  - The scalar product is computed by a specialized function called kernel

$$k(x, y) = (\phi(x) \cdot \phi(y))$$



CLASSIFICATION FUNCTION: THE DUAL FORM

$$h(x) = sgn(\vec{w} \cdot \vec{x} + b) = sgn(\sum_{J=1}^{l} \alpha_j y_j \vec{x_j} \cdot \vec{x} + b)$$

- On the right form, instances only appear in the scalar product
- The ony thing that is needed is the Gram matrix,

$$G = \left(\left\langle \mathbf{x}_i \cdot \mathbf{x}_j \right\rangle \right)_{i,j=1}^{l}$$

i.e. the explicit computation of the scalar product over any pair of training instances $x_1 \ldots x_l$

A KERNELIZED PERCEPTRON

- We can rewrite the decision function of a perceptron by taking into account a kernel:

$$h(x) = sgn(\vec{w} \cdot \Phi(\vec{x}) + b) = sgn(\sum_{J=1}^{l} \alpha_j y_j \Phi(\overrightarrow{x_j}) \cdot \Phi(\vec{x}) + b)$$

$$= sgn(\sum_{J=1}^{l} \alpha_j y_j k(\overrightarrow{x_j}, \vec{x}) + b)$$

- … and during training the on-line adjustment steps become:

$$y_i(\sum_{J=1}^{l} \alpha_j y_j \Phi(\overrightarrow{x_j}) \cdot)\Phi(\overrightarrow{x_i}) + b) = \sum_{J=1}^{l} \alpha_j y_i y_j k(\overrightarrow{x_j}, \overrightarrow{x_i}) + b)$$

KERNELS IN SUPPORT VECTOR MACHINES

$$\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \vec{x_i} \cdot \vec{x_j} + \frac{1}{2C} \vec{\alpha} \cdot \vec{\alpha} - \frac{1}{C} \vec{\alpha} \cdot \vec{\alpha}$$

- By using kernel functions we rewrite the problem as:

- In Soft Margin SVMs we need to

$$\begin{cases} maximize \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \left(k(o_i, o_j) + \frac{1}{C} \delta_{ij} \right) \\ \alpha_i \geq 0, \quad \forall i = 1, .., m \\ \sum_{i=1}^{m} y_i \alpha_i = 0 \end{cases}$$

WHAT MAKES A FUNCTION A KERNEL FUNCTION?

Def. 2.26 *A kernel is a function k, such that* $\forall\ \vec{x},\vec{z} \in X$

$$k(\vec{x},\vec{z}) = \phi(\vec{x}) \cdot \phi(\vec{z})$$

where ϕ *is a mapping from* X *to an (inner product) feature space.*

Only such type of functions support implicit mappings such as

$$\vec{x} = (x_1, \dots, x_n) \in R^n \ \rightarrow\ \Phi(\vec{x}) = (\Phi_1(\vec{x}), \dots, \Phi_m\ (\vec{x})) \in R^m$$

WHAT MAKES A FUNCTION A KERNEL FUNCTION? (2)

Def. B.11 *Eigen Values*
Given a matrix $\boldsymbol{A} \in \mathbb{R}^n \times \mathbb{R}^n$ *, an egeinvalue* λ *and an egeinvector* $\vec{x} \in \mathbb{R}^n - \{\vec{0}\}$ *are such that*

$$\boldsymbol{A}\vec{x} = \lambda\vec{x}$$

Def. B.12 *Symmetric Matrix*
A square matrix $\boldsymbol{A} \in \mathbb{R}^n \times \mathbb{R}^n$ *is symmetric iff* $\boldsymbol{A}_{ij} = \boldsymbol{A}_{ji}$ *for* $i \neq j$ $i = 1, .., m$ *and* $j = 1, .., n$*, i.e. iff* $\boldsymbol{A} = \boldsymbol{A}'$.

Def. B.13 *Positive (Semi-) definite Matrix*
A square matrix $\boldsymbol{A} \in \mathbb{R}^n \times \mathbb{R}^n$ *is said to be positive (semi-) definite if its eigenvalues are all positive (non-negative).*

WHAT MAKES A FUNCTION A KERNEL FUNCTION? (3)

Proposition 2.27 *(Mercer's conditions)*
Let X be a finite input space with $K(\vec{x}, \vec{z})$ a symmetric function on X. Then $K(\vec{x}, \vec{z})$ is a kernel function if and only if the matrix

$$k(\vec{x}, \vec{z}) = \phi(\vec{x}) \cdot \phi(\vec{z})$$

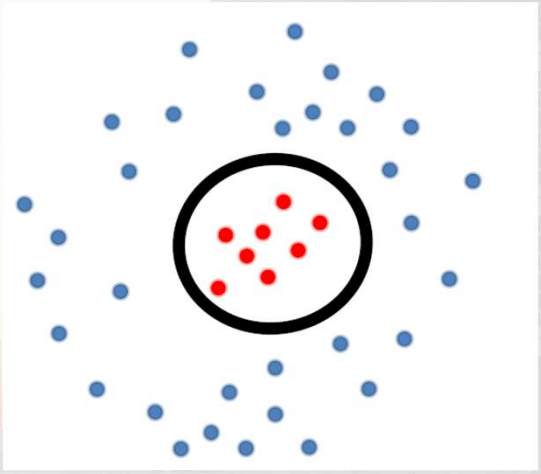
is positive semi-definite (has non-negative eigenvalues).

- IDEA: If the Gram matrix is positive semi-definite then the mapping ϕ, such that *F* is an inner-product space whose scalar product corresponds to the kernel *k(.,.)*, exists
- In *F* the separability should be easier

KERNEL FUNCTIONS: OTHER EXAMPLES

An example of non linear classification

- $g(\vec{x}) = w_1 x_1^2 + w_{21} x_2^2 + w_0$ (ellipsis in $\mathbb{R}^2$)
  - For example circles (of radius *r*)
    - $g(\vec{x}) = x_1^2 + x_2^2 - r^2$
- Can it be made linear in some space?



LINEAR CLASSIFICATION IN THE KERNEL SPACE

- Let try to rewrite the definintion for *g()*

- $g(\vec{x}) = w_1 x_1^2 + w_2 x_2^2 + w_0 = g(\vec{x}) = w_1 \chi_1 + w_2 \chi_2 + w_0 =$

$= w_1 \phi_1(\vec{x}) + w_2 \phi_2(\vec{x}) + w_0$

$(\phi(\vec{x}) = \phi(x_1, x_2) \longmapsto (x_1^2, x_2^2))$

- G is linear in the ϕ-transformed space!

FEATURE SPACES AND KERNELS

- Another example of Kernel
  - The Polynomial kernel $k(x,y) = (x \cdot y)^d$

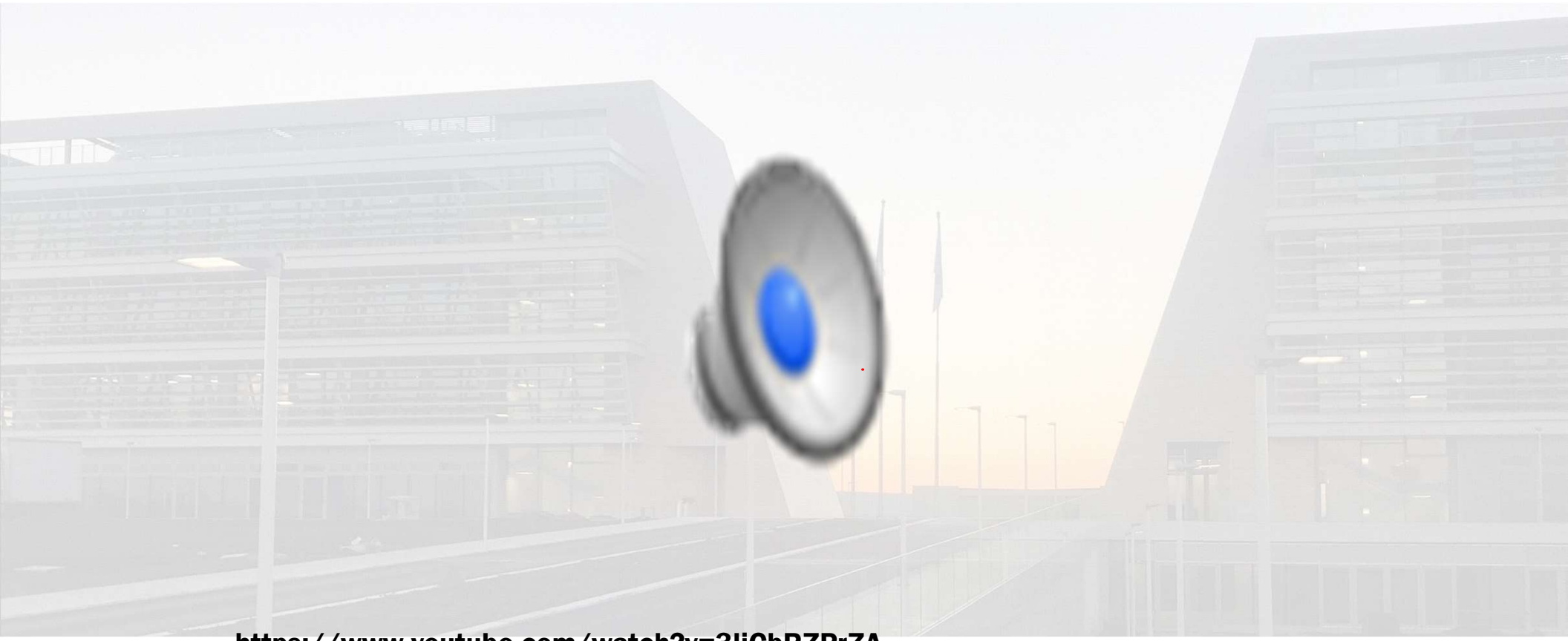
$$x, y \in R^2$$

  - If *d*=2 and

$$(x \cdot y)^2 = \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \right)^2 = \left(\begin{bmatrix} x_1^2 \\ \sqrt{2}\, x_1 x_2 \\ x_2^2 \end{bmatrix} \cdot \begin{bmatrix} y_1^2 \\ \sqrt{2}\, y_1 y_2 \\ y_2^2 \end{bmatrix} \right)$$

$$= (\phi(x) \cdot \phi(y)) = k(x,y)$$

POLYNOMIAL KERNEL



https://www.youtube.com/watch?v=3liCbRZPrZA

POLYNOMIAL KERNEL (*N* DIMENSIONS)

$$
\begin{aligned}
(\vec{x} \cdot \vec{z})^2 &= \left(\sum_{i=1}^{n} x_i z_i\right)^2 &&= \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_i z_i\right) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j z_i z_j &&= \sum_{i,j \in \{1,..,n\}} (x_i x_j)(z_i z_j) \\
&= \sum_{k=1}^{m} X_k Z_k &&= \vec{X} \cdot \vec{Z}
\end{aligned}
$$

GENERAL POLYNOMIAL KERNEL (*N* DIMENSIONS)

$$(\vec{x}\cdot\vec{z}+c)^2=(\sum_{i=1}^{n}x_iz_i+c)^2=(\sum_{i=1}^{n}x_iz_i+c)(\sum_{j=1}^{n}x_iz_i+c)=$$

$$=\sum_{i=1}^{n}\sum_{j=1}^{n}x_ix_jz_iz_j+2c\sum_{i=1}^{n}x_iz_i+c^2=$$

$$=\sum_{i,j\in\{1,..,n\}}(x_ix_j)(z_iz_j)+2c\sum_{i=1}^{n}(\sqrt{2c}x_i)(\sqrt{2c}z_i)+c^2$$

POLYNOMIAL KERNEL AND THE CONJUNCTION OF FEATURES

- The initial vectors can be mapped into a higher dimensional space (*c=1*)

$$\Phi(< x_1, x_2 >) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1)$$

- More expressive, as (x_1x_2) encodes original feature pairs, e.g.

stock+market vs. *downtown+market*

are contributing (when occurring) togheter

- We can smartly compute the scalar product as

$$\Phi(\vec{x}).\ \Phi(\vec{z}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1)\ \ (z_1^2, z_2^2, \sqrt{2}z_1z_2, \sqrt{2}z_1, \sqrt{2}z_2, 1) =$$
$$= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 + 2x_1z_1 + 2x_2z_2 + 1 =$$
$$= (x_1z_1 + x_2z_2 + 1)^2 = \boxed{(\vec{x}.\vec{z} + 1)^2 = K_{p2}\ \ (\vec{x}, \vec{z})}$$

THE ARCHITECTURE OF AN SVM

- It is a non linear classifier (based on a kernel)
- Decision function:

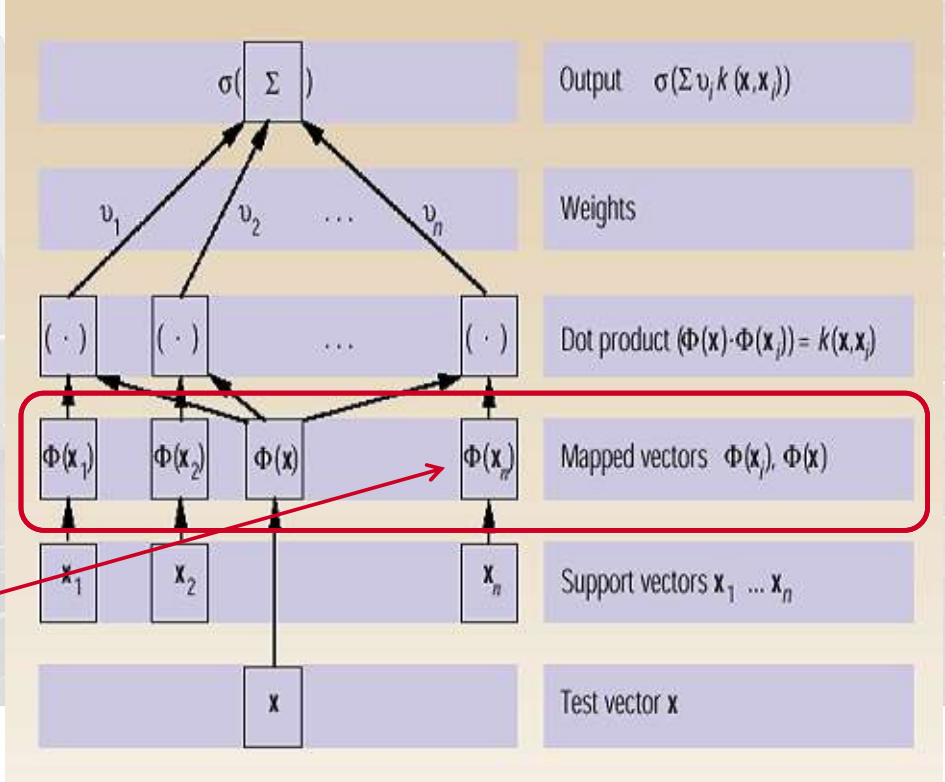
$$f(x) = \text{sgn}\left(\sum_{i=1}^{l} v_i(\varphi(x)\cdot\varphi(x_i)) + b\right) = \text{sgn}\left(\sum_{i=1}^{l} v_i k(x, x_i) + b\right)$$

$\varphi(x_i)$ substitutes every training instance x_i

$v_i = \alpha_i y_i$

v_i are the solutions of the optimization problem

The mapping function is never computed, but is implict in the kernel estimation



OTHER EXAMPLES OF KERNEL FUNCTIONS

- Linear: $k(\vec{x}_i, \vec{x}_j) = \vec{x}_i \cdot \vec{x}_j$
- Polynomial (degree *p*): $k(\vec{x}_i, \vec{x}_j) = (1 + \vec{x}_i \cdot \vec{x}_j)^p$
-
- Gaussian (radial-basis function network):

$$k(\vec{x}_i, \vec{x}_j) = e^{-\frac{\|\vec{x}_i - \vec{x}_j\|^2}{2\sigma^2}}$$

- Perceptron (two stages): $k(\vec{x}_i, \vec{x}_j) = \tanh(\beta_1 + \beta_0 \vec{x}_i \cdot \vec{x}_j)^p$

KERNEL COMBINATION AND NORMALIZATION

- Kernels can be easily combined so that the evidences captured by several kernel functions can contribute to the learning algorithm
  - Linear operations on kernel functions preserve the kernel properties
  - The sum of kernels is a valid kernel
  - The product of kernels is a valid kernel
- We can also normalize the implicit space operating directly only the kernel function

$$\begin{aligned}\hat{K}(s,t) &= \left\langle \hat{\phi}(s) \cdot \hat{\phi}(t) \right\rangle = \left\langle \frac{\phi(s)}{\|\phi(s)\|} \cdot \frac{\phi(t)}{\|\phi(t)\|} \right\rangle \\ &= \frac{1}{\|\phi(s)\| \, \|\phi(t)\|} \langle \phi(s) \cdot \phi(t) \rangle = \frac{K(s,t)}{\sqrt{K(s,s)K(t,t)}}\end{aligned}$$

STRING KERNEL

- Given two strings, the number of matches between their substrings is computed
- E.g. *Bank* and *Rank*
  - *B, a, n, k, Ba, Ban, Bank, an, ank, nk*
  - *R, a , n , k, Ra, Ran, Rank, an, ank, nk*
- String kernel over sentences and texts
- Huge space but there are efficient algorithms
  - Lodhi, Huma; Saunders, Craig; Shawe-Taylor, John; Cristianini, Nello; Watkins, Chris (2002). "*Text classification using string kernels*". Journal of Machine Learning Research: 419–444.

STRING KERNEL

- A function that give two strings s and *t* is able to compute a real number *k(s,t)* such that
  - two vectors exist $\vec{s}$ and $\vec{t}$
  - $\vec{s}$ and $\vec{t}$ are unique for s and *t*
  - (the vectors represents strings by ***embedding*** their crucial properties!!)
  - k(s,t) = $\vec{s} \times \vec{t}$
- We will see how vectors $\vec{s}$ and $\vec{t}$ are defined in $\mathbb{R}^{\infty}$, as the numer of strings of arbitrary length over an alphabet is infinite

- IDEA: Define a space whereas each substring is a dimension

KERNEL TRA *BANK* E *RANK*

B, a, n, k, Ba, Ban, Bank, an, ank, nk, Bn, Bnk, Bk and ak are the substrings of *Bank*.

R, a, n, k, Ra, Ran, Rank, an, ank, nk, Rn, Rnk, Rk and ak are the substrings of *Rank*.

ϕ

ϕ(Bank)= (λ , 0, λ , λ , λ , λ^2 , λ^2, λ^3 , 0 , λ^4 , 0 , λ^2 , λ^3 , λ^3 , ...

ϕ(Rank)= (0 , λ, λ , λ , λ , 0 , 0 , 0 , λ^3 , 0 , λ^4 , λ^2 , λ^3 , λ^3 , ...

B , R, a, n , k, Ba, Ra, Ban, Ran, Bank, Rank, an, ank , ak ...

- Common substrings:
  - *a*, *n*, *k*, *an*, *ank*, *nk*, *ak*
- Notice how these are the same subsequences as between
  - *Schrianak* and *Rank*

FORMALLY …

Sottosequenza di indici ordinati e non contigui di (*1, …* |s|)

$$s = s_1, .., s_{|s|}$$

$$\vec{I} = (i_1, ..., i_{|u|}) \qquad u = s[\vec{I}]$$, substring of s defined by $\vec{I}$

$$\phi_u(s) = \sum_{\vec{I}:u=s[\vec{I}]} \lambda^{l(\vec{I})}, \text{ con } l(\vec{I}) = i_{|u|} - i_1 + 1$$

$$K(s,t) = \sum_{u \in \Sigma^*} \phi_u(s) \cdot \phi_u(t) = \sum_{u \in \Sigma^*} \sum_{\vec{I}:u=s[\vec{I}]} \lambda^{l(\vec{I})} \sum_{\vec{J}:u=t[\vec{J}]} \lambda^{l(\vec{J})} =$$

$$= \sum_{u \in \Sigma^*} \sum_{\vec{I}:u=s[\vec{I}]} \sum_{\vec{J}:u=t[\vec{J}]} \lambda^{l(\vec{I})+l(\vec{J})}, \text{ con } \Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$$

AN EXAMPLE OF STRING KERNEL COMPUTATION

- $\phi_{\mathrm{a}}(\mathrm{Bank}) = \phi_{\mathrm{a}}(\mathrm{Rank}) = \lambda^{(i_1 - i_1 + 1)} = \lambda^{(2-2+1)} = \lambda,$
- $\phi_{\mathrm{n}}(\mathrm{Bank}) = \phi_{\mathrm{n}}(\mathrm{Rank}) = \lambda^{(i_1 - i_1 + 1)} = \lambda^{(3-3+1)} = \lambda,$
- $\phi_{\mathrm{k}}(\mathrm{Bank}) = \phi_{\mathrm{k}}(\mathrm{Rank}) = \lambda^{(i_1 - i_1 + 1)} = \lambda^{(4-4+1)} = \lambda,$
- $\phi_{\mathrm{an}}(\mathrm{Bank}) = \phi_{\mathrm{an}}(\mathrm{Rank}) = \lambda^{(i_1 - i_2 + 1)} = \lambda^{(3-2+1)} = \lambda^2,$
- $\phi_{\mathrm{ank}}(\mathrm{Bank}) = \phi_{\mathrm{ank}}(\mathrm{Rank}) = \lambda^{(i_1 - i_3 + 1)} = \lambda^{(4-2+1)} = \lambda^3,$

$\phi_{\mathrm{nk}}(\mathrm{Bank}) = \phi_{\mathrm{nk}}(\mathrm{Rank}) = \lambda^{(i_1 - i_2 + 1)} = \lambda^{(4-3+1)} = \lambda^2,$

$\phi_{\mathrm{ak}}(\mathrm{Bank}) = \phi_{\mathrm{ak}}(\mathrm{Rank}) = \lambda^{(i_1 - i_2 + 1)} = \lambda^{(4-2+1)} = \lambda^3.$

It follows that $K(\mathrm{Bank}, \mathrm{Rank}) = (\lambda, \lambda, \lambda, \lambda^2, \lambda^3, \lambda^2, \lambda^3) \cdot (\lambda, \lambda, \lambda, \lambda^2, \lambda^3, \lambda^2, \lambda^3)$
$= 3\lambda^2 + 2\lambda^4 + 2\lambda^6.$