KERNEL-BASED LEARNING

WM&R a.a. 2020/21

D. Croce, R. Basili (A. Moschitti)
Università di Roma “Tor Vergata”
basili@info.uniroma2.it

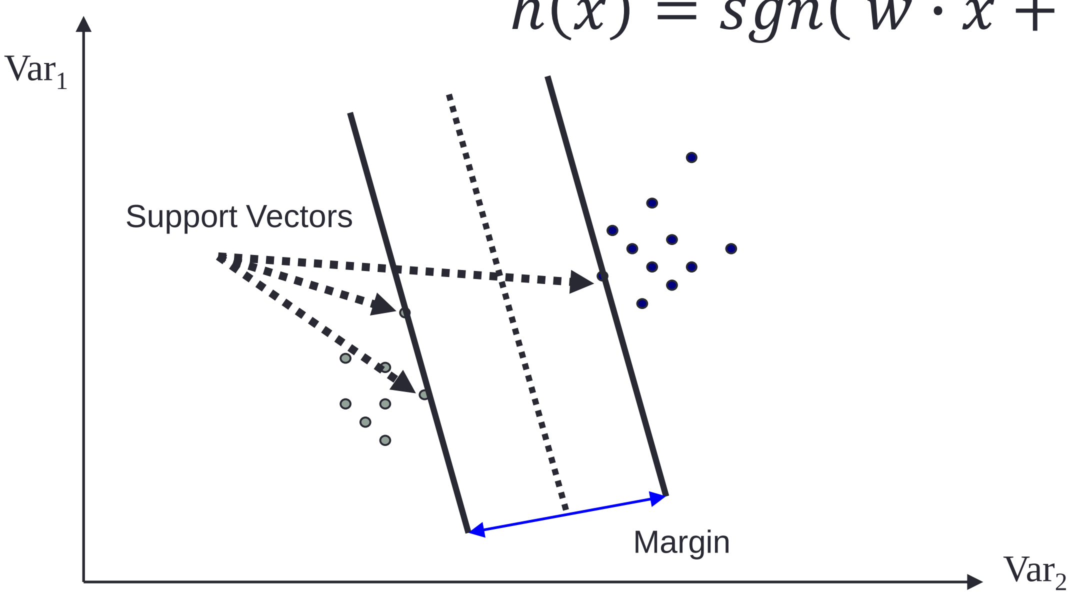
Outline

- Metodi Kernel
  - Motivazioni
  - Esempio
- Kernel standard
  - Polynomial kernel
  - String Kernel
- Introduzione a metodi Kernel *avanzati*
  - Tree kernels

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

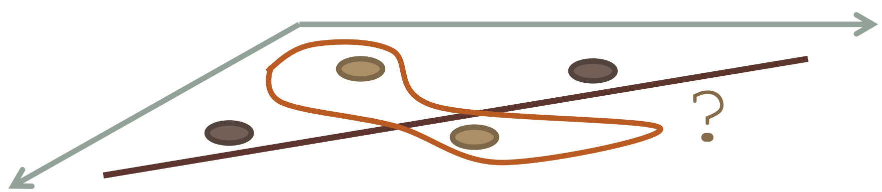
$$h(x) = sgn(\vec{w} \cdot \vec{x} + b) = sgn(\underbrace{\sum_{J=1}^{l} \alpha_j y_j \boxed{\vec{x_j} \cdot \vec{x}}}_{\text{Support Vectors}} + b)$$



Only the dot product is required

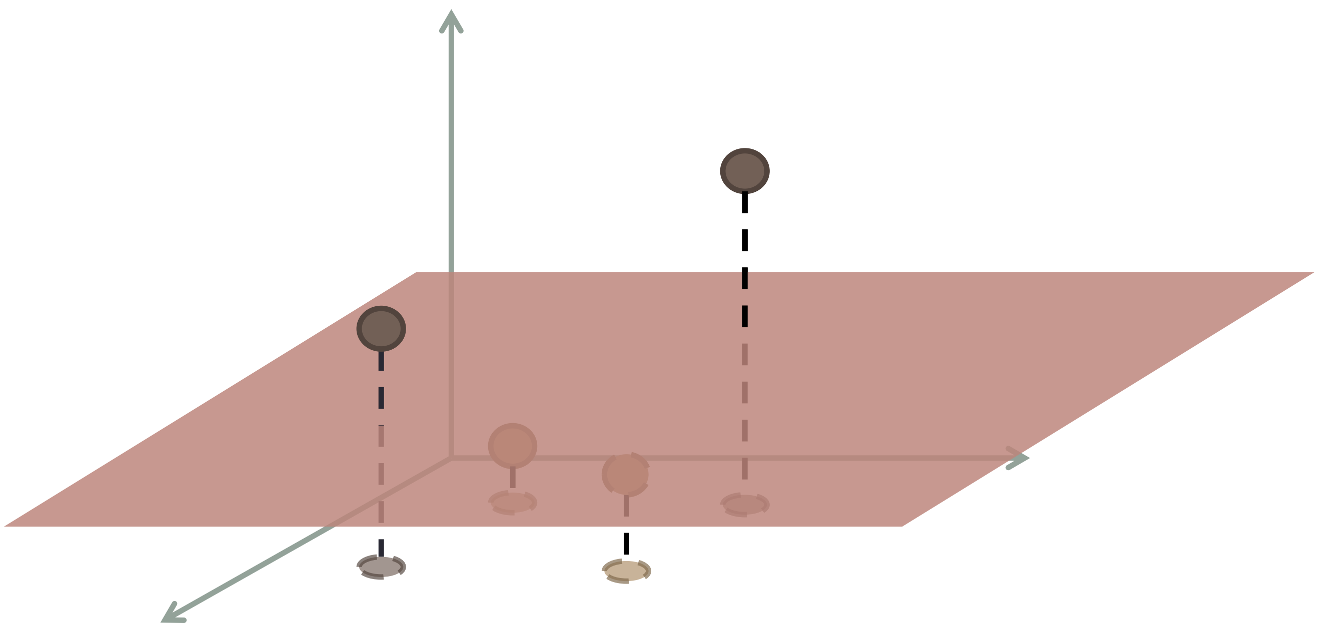
Linear classifiers and separability

- 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)



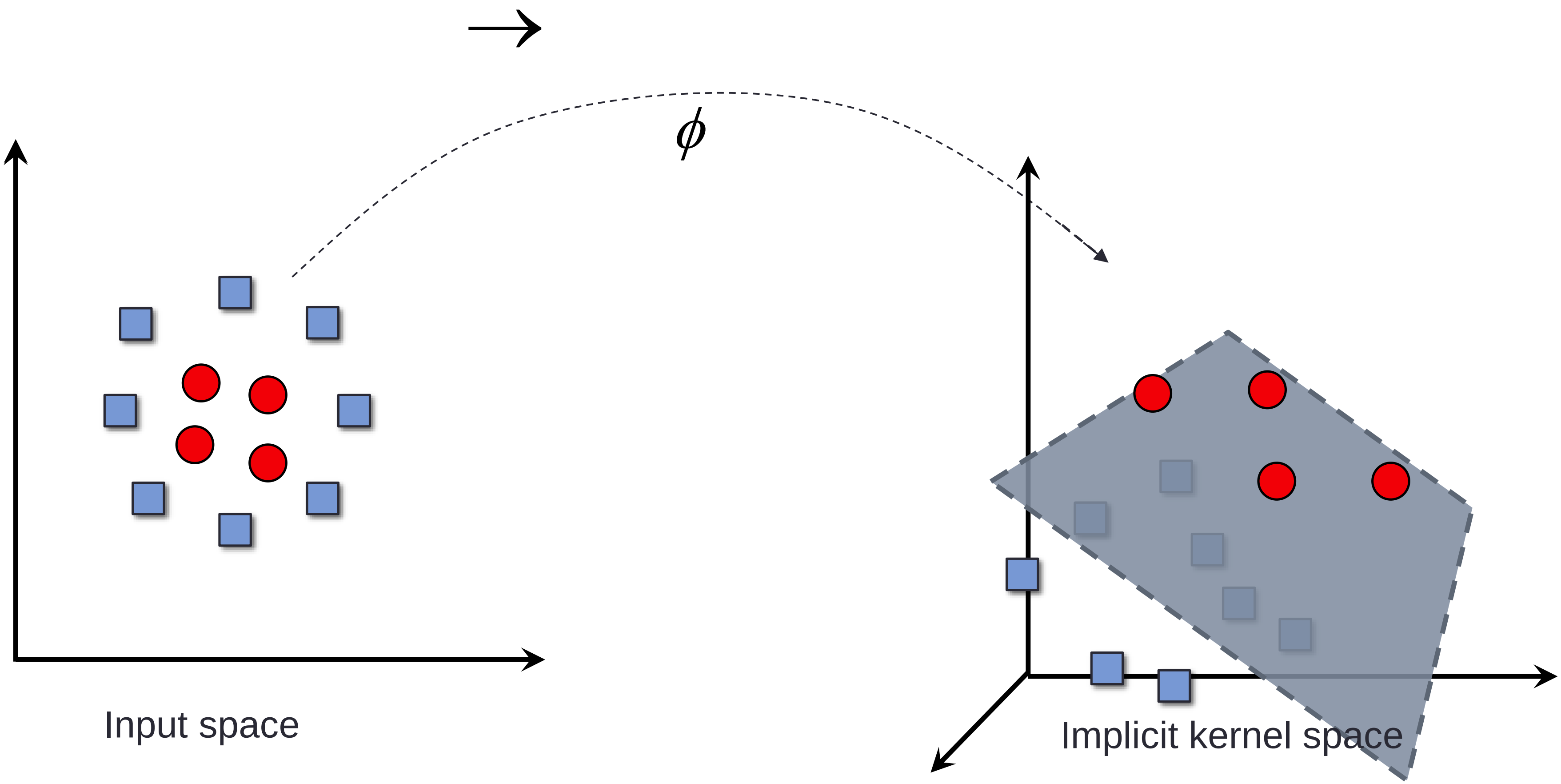
The kernel function

- 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 $\boxed{f(m_1, m_2, r) < f_a}$

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()*:

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

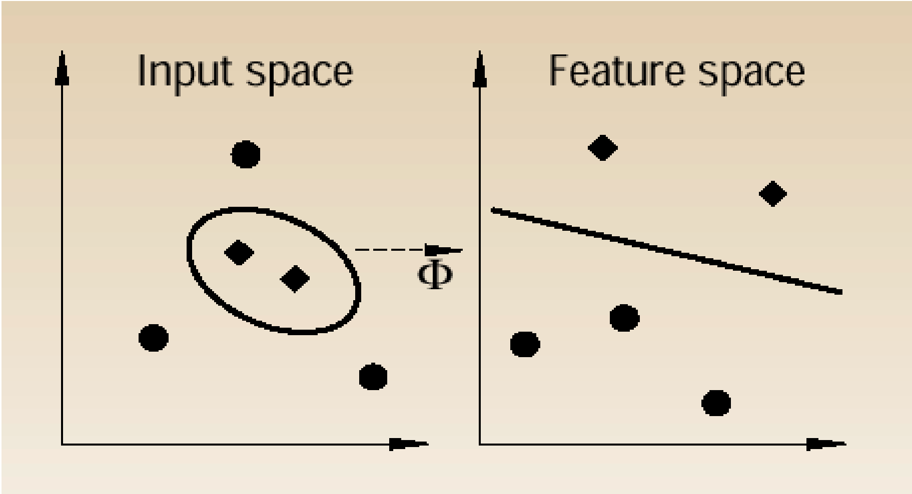
$$(1,1,-2,-1) \cdot (\ln m_1, \ln m_2, \ln r, \ln f_a) + \ln C = 0,$$

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

$$\phi = R^N \rightarrow F$$



- The kernel function
  - The evaluation require the computation of the scalar product over the trasformed vectors $\phi(x)$ but not the feature vectors themselves
  - The scalr product is computed by a specialized function called kernel

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

Classification function: the dual form

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

- 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 … x_l

A kernelized perceptron

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

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

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

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

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

Kernels in Support Vector Machines

- In Soft Margin SVMs we need to maximize :

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

- By using kernel functions we rewrite the problem as:

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

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

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

What makes a function a kernel function? (2)

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

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

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

Def. B.13 *Positive (Semi-) definite Matrix*
A square matrix $\mathbf{A} \in \mathbb{R}^n \times \mathbb{R}^n$ *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

Feature Spaces and Kernels

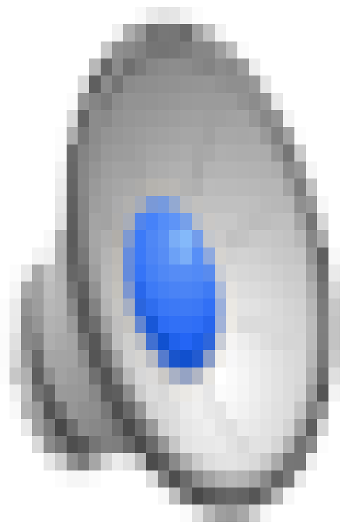
- An example of Kernel
  - The Polynomial kernel
    - If *d=2* and $k(x,y) = (x \cdot y)^d$

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

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

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

Polynomial kernel



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

Polynomial Kernel (*n* dimensions)

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

General Polynomial Kernel (*n* dimensions)

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

$$= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j z_i z_j + 2c\sum_{i=1}^{n} x_i z_i + c^2 =$$

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

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

$$\Phi(\vec{x}) \times \Phi(\vec{z}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1) \times (z_1^2, z_2^2, \sqrt{2}z_1z_2, \sqrt{2}z_1, \sqrt{2}z_2, 1) =$$

$$= x_1^2 z_1^2 + x_2^2 z_2^2 + 2x_1x_2z_1z_2 + 2x_1z_1 + 2x_2z_2 + 1 =$$

$$= (x_1z_1 + x_2z_2 + 1)^2 = \boxed{(\vec{x} \times \vec{z} + 1)^2 = K_{p2}(\vec{x}, \vec{z})}$$

The Architecture of an SVM

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

$$f(x) = \operatorname{sgn}\left(\sum_{i=1}^{l} v_i(\phi(x)\cdot\phi(x_i)) + b\right)$$

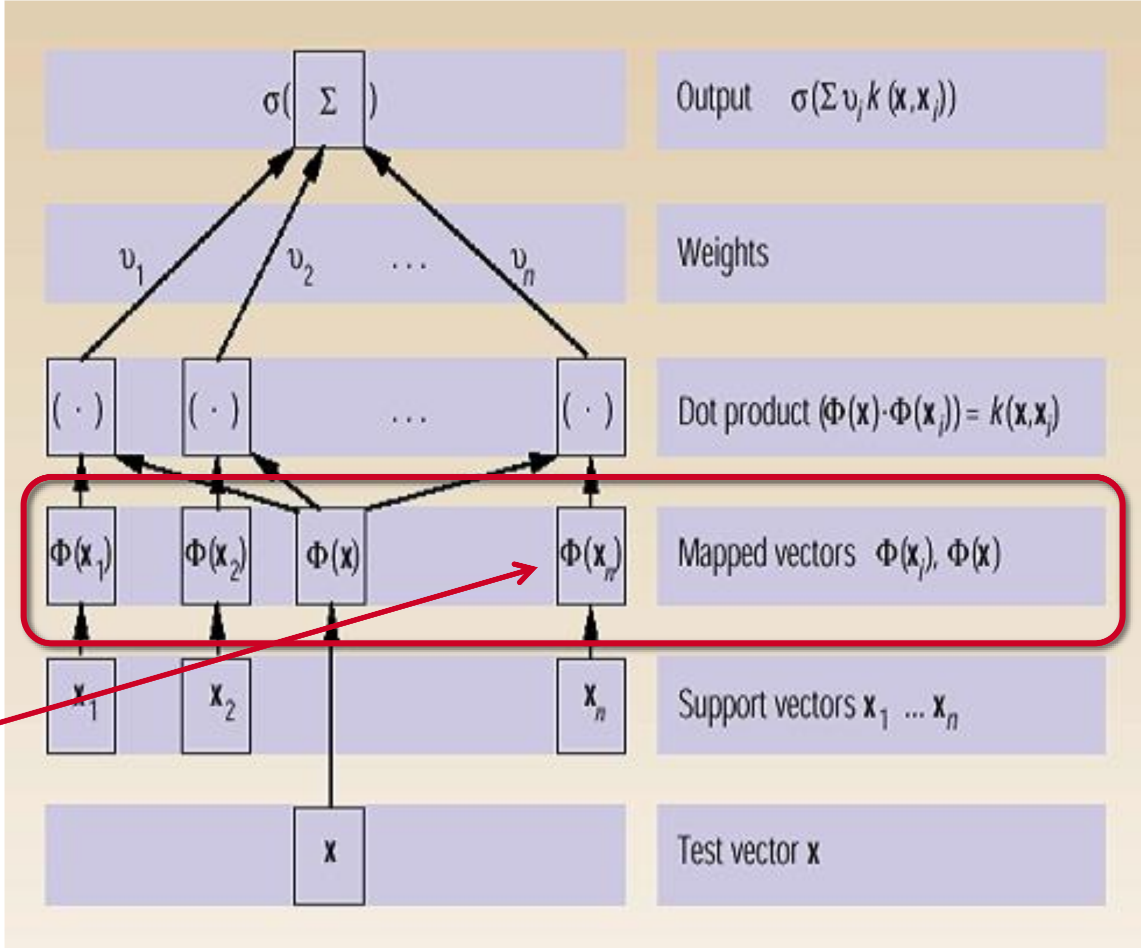
$$= \operatorname{sgn}\left(\sum_{i=1}^{l} v_i k(x, x_i) + b\right)$$

$\phi(x_i)$ *substitutes every training instamce* 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



Esempi di Funzioni Kernel

- Lineare: $k(\vec{x}_i, \vec{x}_j) = \vec{x}_i \cdot \vec{x}_j$

- Polinomiale potenza di p: $k(\vec{x}_i, \vec{x}_j) = (1 + \vec{x}_i \cdot \vec{x}_j)^p$
-
- Gaussiana (radial-basis function network):

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

- Percettrone a due stadi:

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

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 $Rank$.

$$\Big\Downarrow \phi$$

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 <u>ordinati</u> e <u>non contigui</u> di (1, ... |s|)

$$s = s_1, \ldots, s_{|s|}$$

$$\vec{I} = \boxed{(i_1, \ldots, i_{|u|})}$$

$u = s[\vec{I}]$, 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 $K(\mathrm{Bank}, \mathrm{Rank}) = (\lambda, \lambda, \lambda, \lambda^2, \lambda^3, \lambda^2, \lambda^3) \cdot (\lambda, \lambda, \lambda, \lambda^2, \lambda^3, \lambda^2, \lambda^3)$
$= 3\lambda^2 + 2\lambda^4 + 2\lambda^6.$

Tree Kernels

- String kernels adopt a structured approach to kernel estimation and are very useful in NLP and Web Mining tasks
- However, what has been defined over sequences can be profitably exploited also in the treatment of more complex structures
  - Trees whose parent relationship determine subsequences in terms of
    - Multiple paths from the root to the leaves
    - Ordered sets of children (i.e. sequences of immediately dominated nodes) of every node in the tree
  - Graphs, whose structure can be captured by several trees (subgraphs) and thus characterized by multiple subsequences

Tree kernels

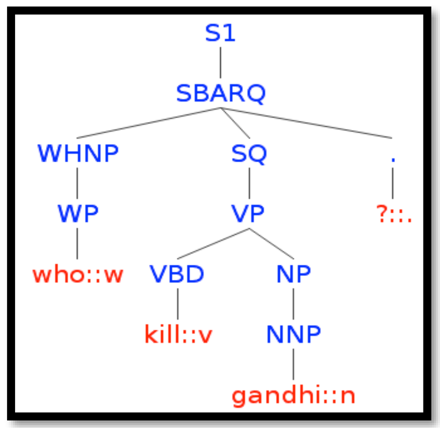
- Applications are related to **text processing** tasks such as
  - Syntactic parsing, when SVM classification is useful to select the best parse tree among multiple legal grammatical interpretations
  - Question Classification, where SVM classification is applied to the recognition of the target of a question (e.g. a person such as in "*Who is the inventor of the light?*" vs. a place as in "*Where is Taji Mahal?*"

or to **pattern recognition** (e.g. in bioinformatics the classification of protein structures)

Tree Kernels

Modeling syntax in Natural Language learning task is complex, e.g.

- Question Classification
- Semantic role relations within predicate argument structures and



Tree kernels are natural way to exploit syntactic information from sentence parse trees

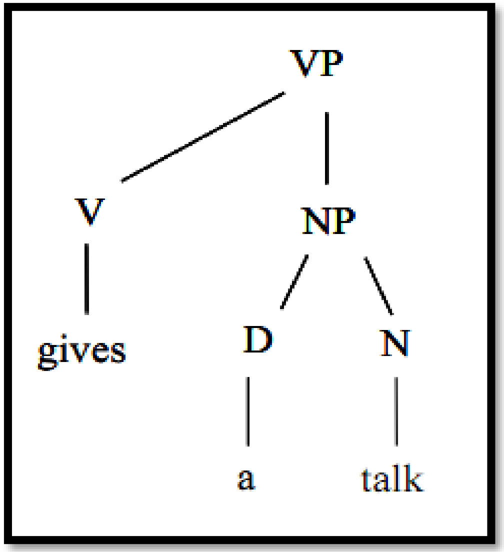
- useful to engineer novel and complex features.

Tree structures and natural language

- **PARSING**: Breaking down a text into its component parts of speech (according to a formal grammar) with an explanation of the form, function, and syntactic relationship of each part

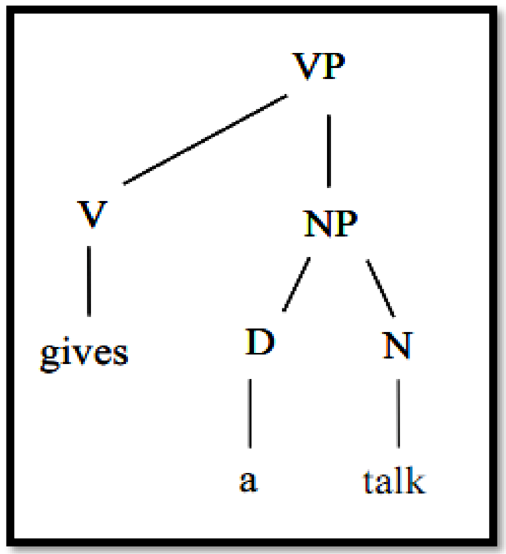
- INPUT: *gives a talk*

- Output : a ***costituency*** tree



Chomsky, N. 1957. Syntactic Structures. The Hague/Paris: Mouton.

The Collins and Duffy’s Tree Kernel

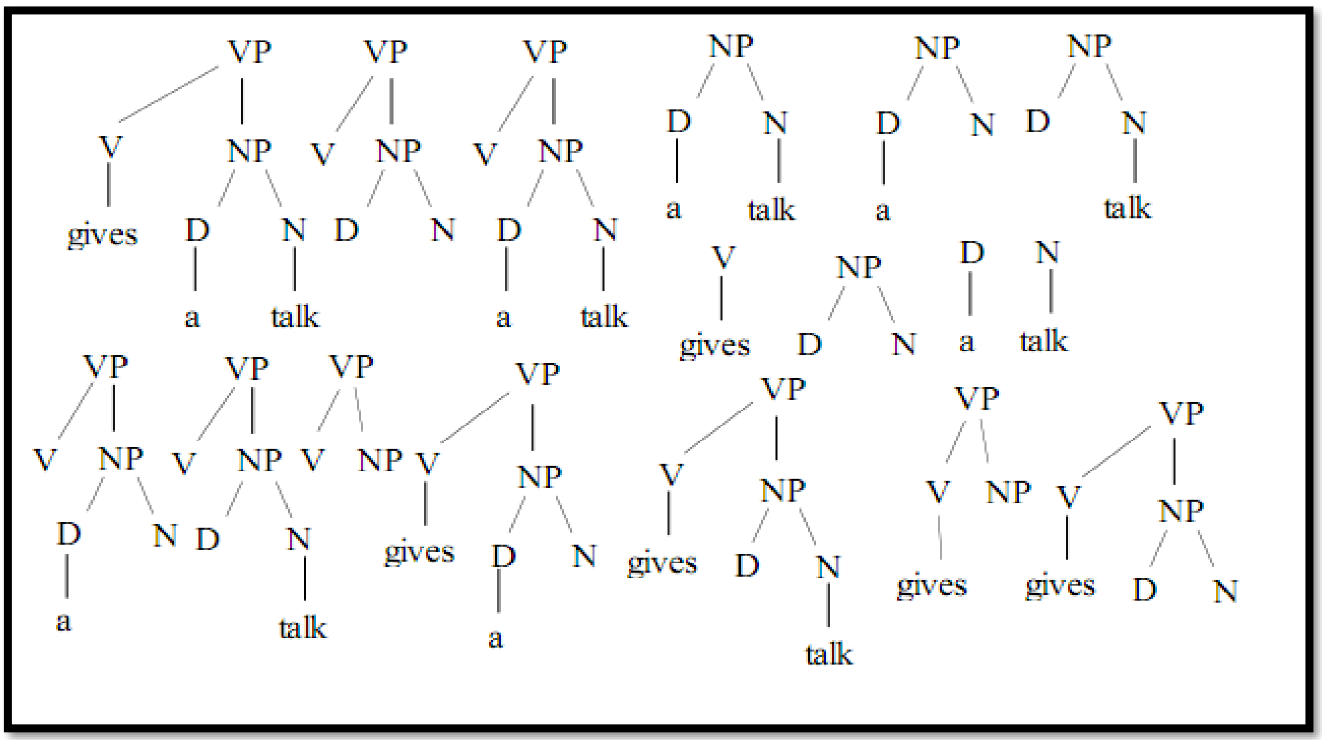


Given a costituency tree

The overall fragment set

We can explode the syntactic tree in *all* ***syntactically motivated*** fragments

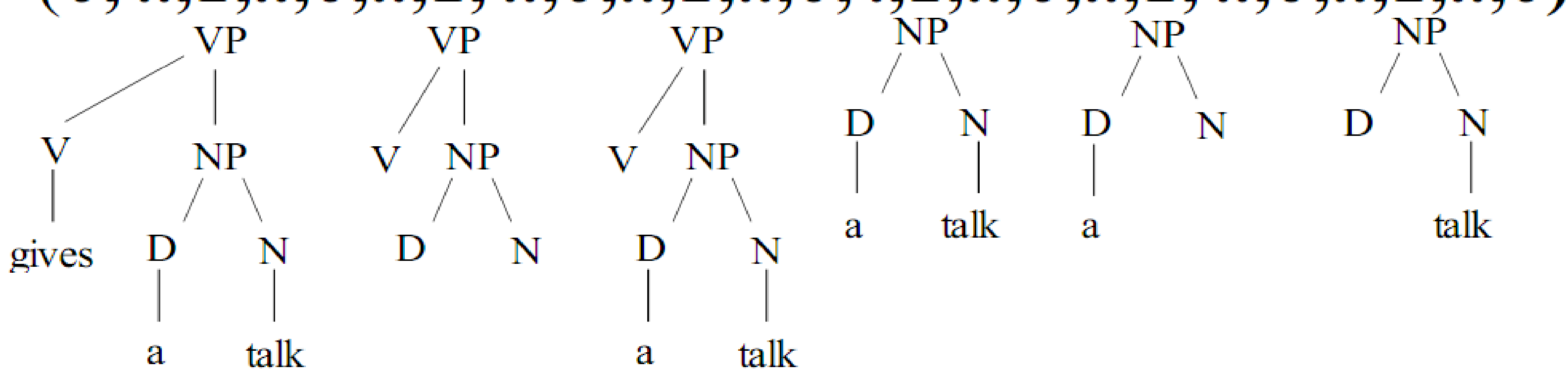
- For each node the production rules must be respected, i.e. we can remove “0 or all children at a time”
- It is also known as Syntactic Tree Kernel



Explicit feature space

Can we build a feature vector accounting on all this information?

$$\vec{x} = (\mathbf{0, .., 1, .., 0, .., 1, .., 0, .., 1, .., 0, .., 1, .., 0, .., 1, .., 0, .., 1, .., 0})$$



$\vec{x}_1 \cdot \vec{x}_2$ counts the number of common substructures

Implicit Representation

Can we estimate the tree kernel in an implicit space?

- We can implicitly count the number of common subtrees
- We prevent to define feature vectors that consider ALL POSSIBLE SUBTREES, i.e. thousand of features
- The final model will not contain feature vectors, but TREES

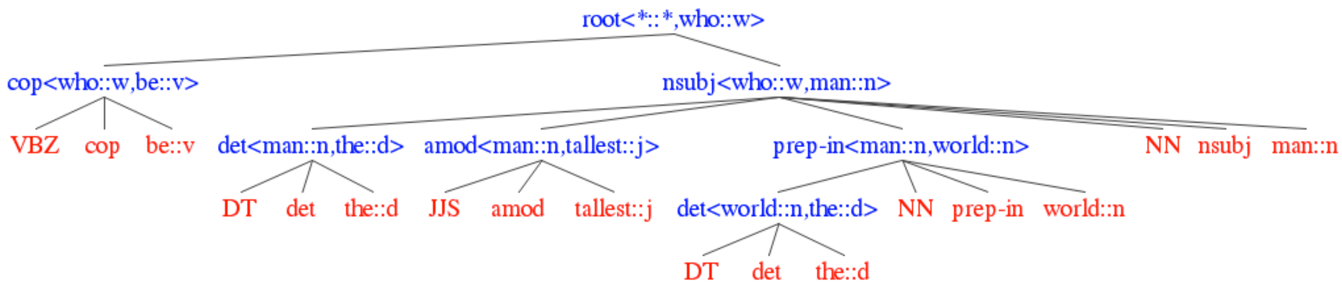
$$\vec{x}_1 \cdot \vec{x}_2 = \phi(T_1) \cdot \phi(T_2) = K(T_1, T_2) =$$
$$= \sum_{n_1 \in T_1} \sum_{n_2 \in T_2} \Delta(n_1, n_2)$$

[Collins and Duffy, ACL 2002] evaluate Δ in $O(n^2)$:

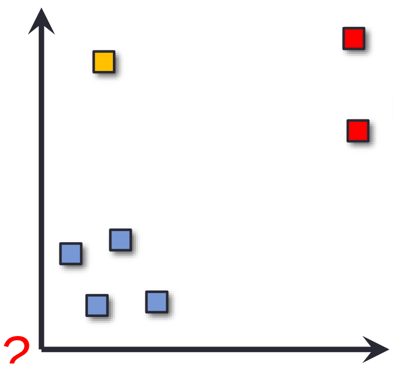
$$\Delta(n_1, n_2) = 0, \ \textbf{if the productions are different else}$$
$$\Delta(n_1, n_2) = 1, \ \textbf{if pre-terminals else}$$
$$\Delta(n_1, n_2) = \prod_{j=1}^{nc(n_1)} (1 + \Delta(ch(n_1, j), ch(n_2, j)))$$

Tree kernels are … embedding tools

- Semantic Tree Kernels allows generating vectors that reflect syntactic/semantic information of sentences
  - *Who is the tallest man in the world?*



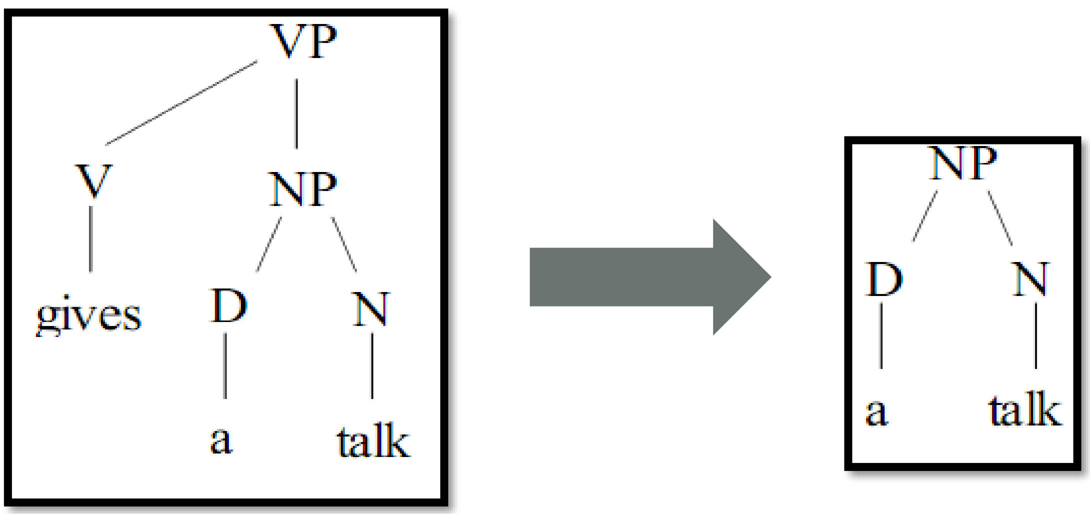
- Which most similar sentences/trees/vectors?
  - *Who is the richest woman in the world?*
  - *Who is the richest person in the world?*
  - *Who is the fastest swimmer in the world?*
  - *Who was murdered yesterday by the terrorist group?*
  - ….



Weighting in grammatical tree kernels

In the kernel estimation different subtrees are taken in account different times

- Es: in the following trees, one fragment will contribute twice to the overall kernel



Weighting

- A decay factor can be used, so the contribution of the embedded trees is reduced.
- The normalization of Tree Kernel estimation corresponds to the normalization of the explicit feature vector

Decay factor

$$\Delta(n_1, n_2) = \lambda, \quad \textbf{if pre-terminals else}$$

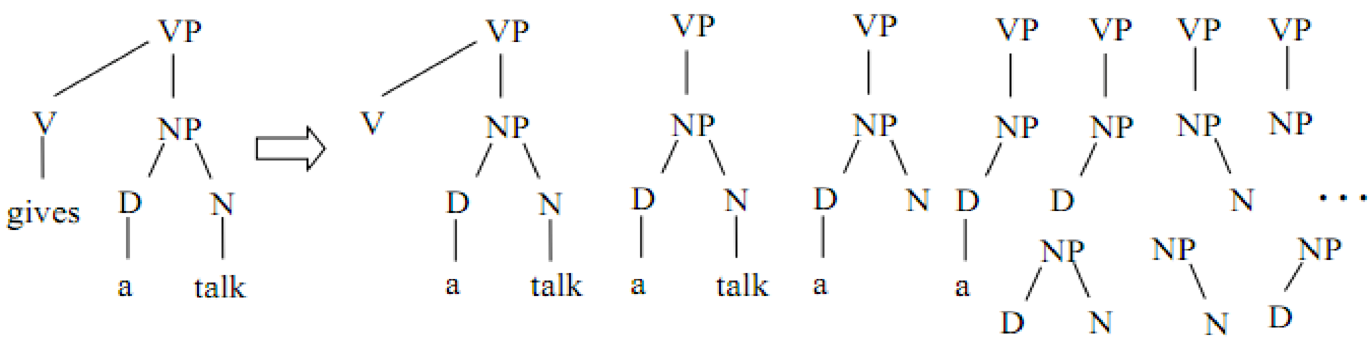
$$\Delta(n_1, n_2) = \lambda \prod_{j=1}^{nc(n_1)} (1 + \Delta(ch(n_1, j), ch(n_2, j)))$$

Normalization $$K'(T_1, T_2) = \frac{K(T_1, T_2)}{\sqrt{K(T_1, T_1) \times K(T_2, T_2)}}$$

Partial Tree [Moschitti,2006]

- A Syntactic Tree satisfies completely a grammar rule, i.e. the constraint is “*remove 0 or all children at a time*”.

- Partial Tree Kernel (PTK) relaxes such constraint we get more general substructures
  - It allows gaps in the production rules in the same fashion of the sequence kernel



Partial Tree Kernel

- if the node labels of n_1 and n_2 are different then $\Delta(n_1, n_2) = 0$;
- else

$$\Delta(n_1, n_2) = 1 + \sum_{\vec{J}_1, \vec{J}_2, l(\vec{J}_1) = l(\vec{J}_2)} \prod_{i=1}^{l(\vec{J}_1)} \Delta(c_{n_1}[\vec{J}_{1i}], c_{n_2}[\vec{J}_{2i}])$$

- By adding two decay factors we obtain:

$$\mu\Big(\lambda^2 + \sum_{\vec{J}_1, \vec{J}_2, l(\vec{J}_1) = l(\vec{J}_2)} \lambda^{d(\vec{J}_1) + d(\vec{J}_2)} \prod_{i=1}^{l(\vec{J}_1)} \Delta(c_{n_1}[\vec{J}_{1i}], c_{n_2}[\vec{J}_{2i}])\Big)$$

Kernel Combination and normalization

- Kernels can be easily combined so that the evidences captured by several kernel functions can contribute to the learning algorithm
  - 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

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

Summary

- The dual form of the SVM optimization problem ONLY depends on the scalar product between training examples and NOT from their explicit vector representation (likewise the perceptron)
- This suggests to exploit this property in order to:
  - Define efficient functions able to compute the scalar product out from the original representation (i.e. from the input space)
  - Exploit more complex representations (i.e. more expressive feature spaces) in implicit way
- This corresponds to search the model in feature spaces able to:
  - Preserve the mathematical properties sufficient to guarantee convergence (i.e. the minimization of the expected error)
  - Support training and classification by a limited complexity (e.g. no need to build large dimensional representations of input instances)

Summary (2)

- In order for a function k(.,.) to be a valid kernel, its correspondin Gram matrix mast be positive semi-definite
- In practice, such property is verified empirically over the training datasets
- In this unit, the following kernel funcrion have been introduced as they can be very effective in Web Mining problems:
  - Base kernels (for example, polynomial kernel polinomiali of degree 2)
  - Task dependent kernels that dipenden on the structura of a learning task:
    - String (Sequence) kernels
    - Tree kernels
- We will explore semantic kernels (e.g. latent semantic kernels) later in the course

References

- Kernel Methods for Pattern Analysis, John Shawe-Taylor & Nello Cristianini - Cambridge University Press, 2004
- Haussler, D. (1999). Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, UC Santa Cruz
- Lodhi, Huma; Saunders, Craig; Shawe-Taylor, John; Cristianini, Nello; Watkins, Chris (2002). "Text classification using string kernels". Journal of Machine Learning Research: 419–444.
- Roberto Basili, Marco Cammisa and Alessandro Moschitti, Effective use of wordnet semantics via kernel-based learning. In Proceedings of the 9th Conference on Computational Natural Language Learning (CoNLL 2005), Ann Arbor(MI), USA, 2005
- Building Semantic Kernels for Text Classification using Wikipedia, Pu Wang and Carlotta Domeniconi, Department of Computer Science, George Mason University