

KERNEL-BASED LEARNING

WM&R a.a. 2015/16

R. Basili (A. Moschitti)
Università di Roma "Tor Vergata"
`basili@info.uniroma2.it`

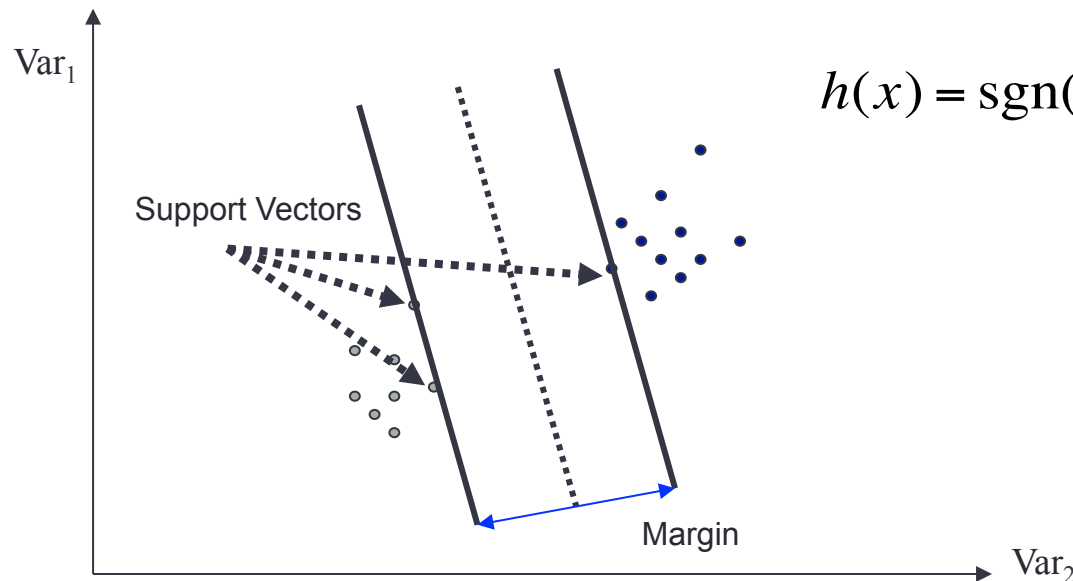
Outline

- 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

Only the dot product is required

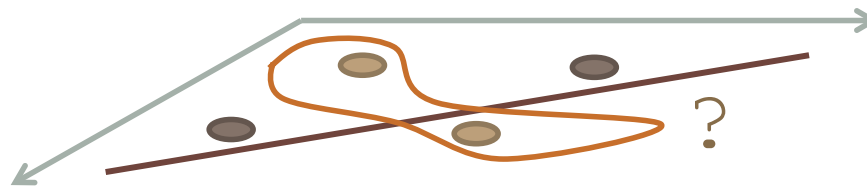


$$h(x) = \text{sgn}(\vec{w} \cdot \vec{x} + b) = \text{sgn}\left(\underbrace{\sum_{j=1..l} \alpha_j y_j \vec{x}_j \cdot \vec{x}}_{\text{Support Vectors}} + b\right)$$

Support
Vectors

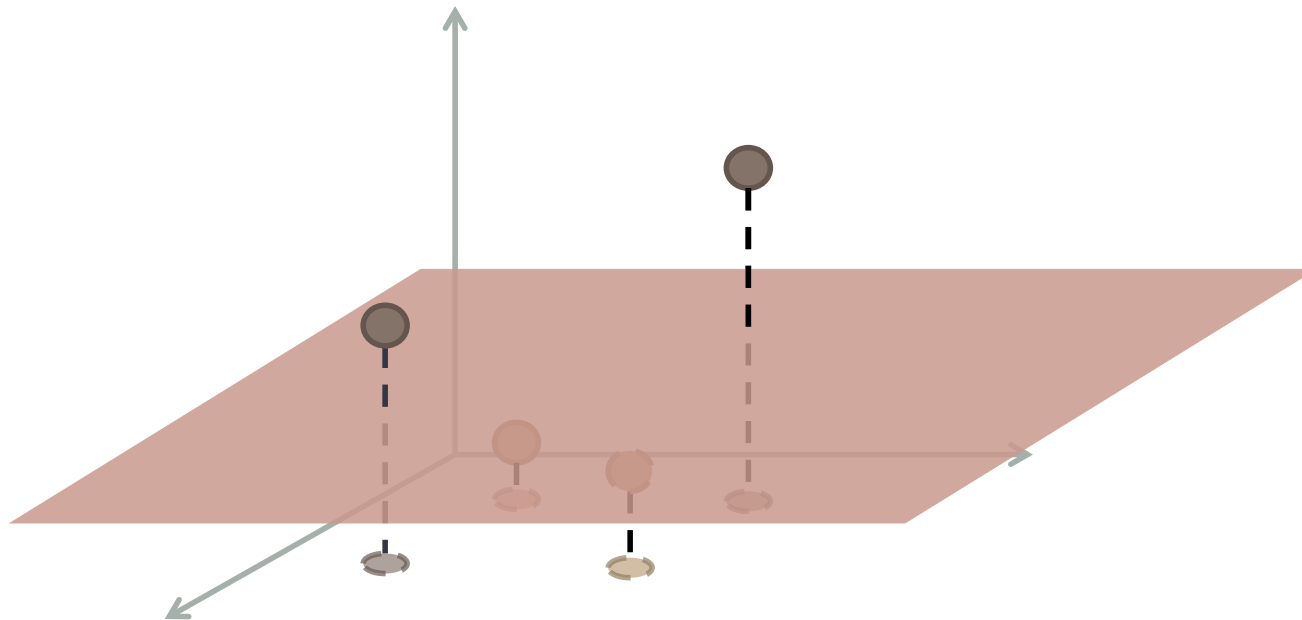
Linear classifiers and separability

- In a \mathbb{R}^2 space, 3 points 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



La funzione Kernel

- Il learning dipende solo dal prodotto scalare dei vettori di esempio
- Quindi dipende dalla Gram-matrix. In generale si definisce kernel la funzione:

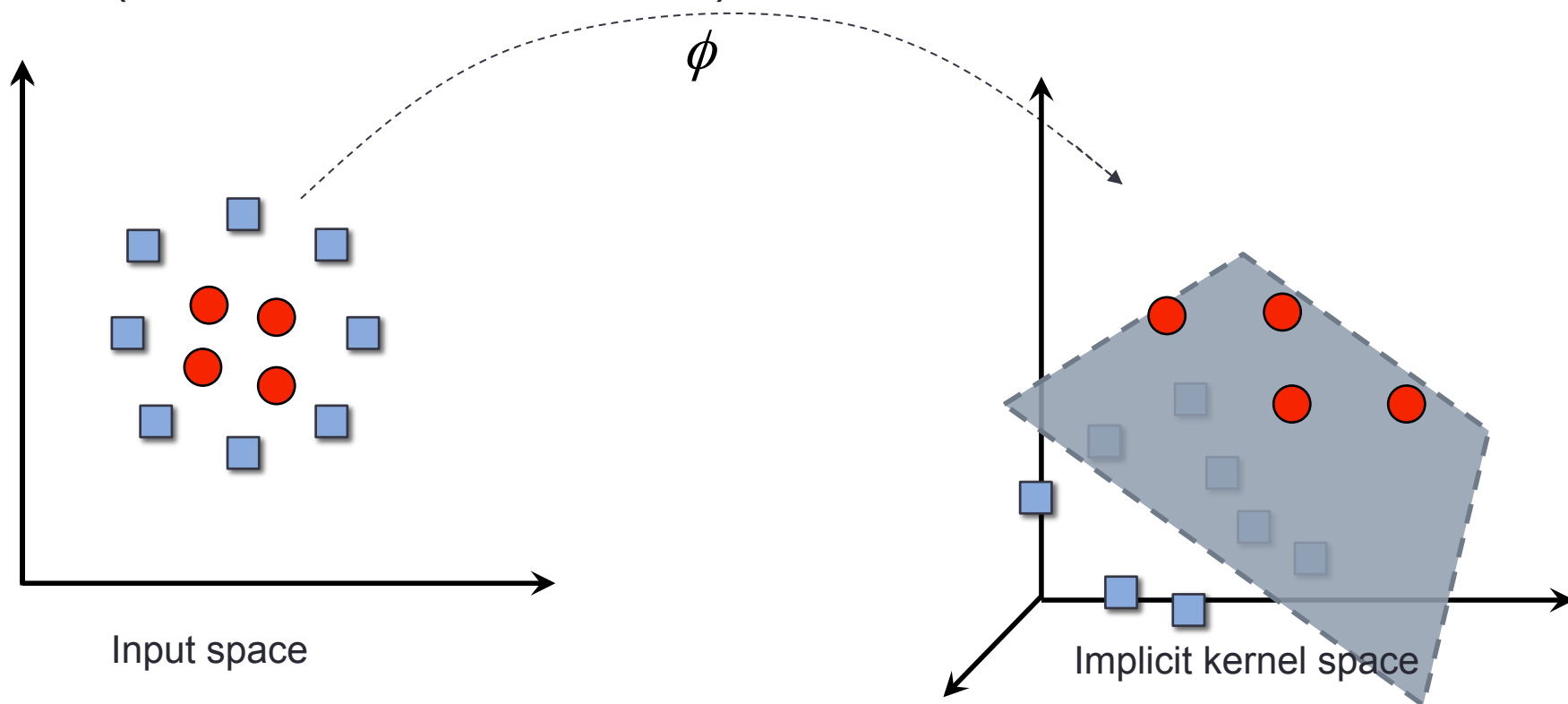
$$K(\vec{z}, \vec{x}) = \phi(\vec{z}) \cdot \phi(\vec{x})$$

- La funzione kernel produce il risultato a partire dagli oggetti iniziali
- Quando la mappatura ϕ è l'identità abbiamo l'usuale prodotto scalare.
- Gli esempi compaiono nell'algoritmo di learning (ad es. il perceptrone) solo attraverso i loro contributi al kernel

Primo Vantaggio: rendere linearmente separabili gli esempi

- Mappare i dati in uno Spazio di Feature dove sono linearmente separabili $\vec{x} \rightarrow \phi(\vec{x})$

(i.e. attributi \rightarrow feature)



Esempio di una funzione di mappatura

- Due masse m_1 e m_2 , una vincolata
- Applico una forza f_a alla massa m_1
- Esperimenti
 - Features m_1 , m_2 e f_a
- Supponiamo di volere apprendere **quando m_1 si allontana da m_2**
- Considerando la legge gravitazionale di Newton

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

- Dobbiamo determinare se $f(m_1, m_2, r) < f_a$

Esempio di una funzione di mappatura

$$\vec{x} = (x_1, \dots, x_n) \rightarrow \phi(\vec{x}) = (\phi_1(\vec{x}), \dots, \phi_n(\vec{x}))$$

- Non esprimibile linearmente, quindi cambio spazio

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

- Poiché

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

- Allora l'iperpiano è la funzione richiesta

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

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

posso decidere senza errore se le masse si avvicinano o si allontanano

Feature Spaces and Kernels

- Feature Space

- Lo spazio di input è mappato in un nuovo spazio dotato di prodotto scalare F (detto *feature space*) attraverso una trasformazione (non lineare)

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

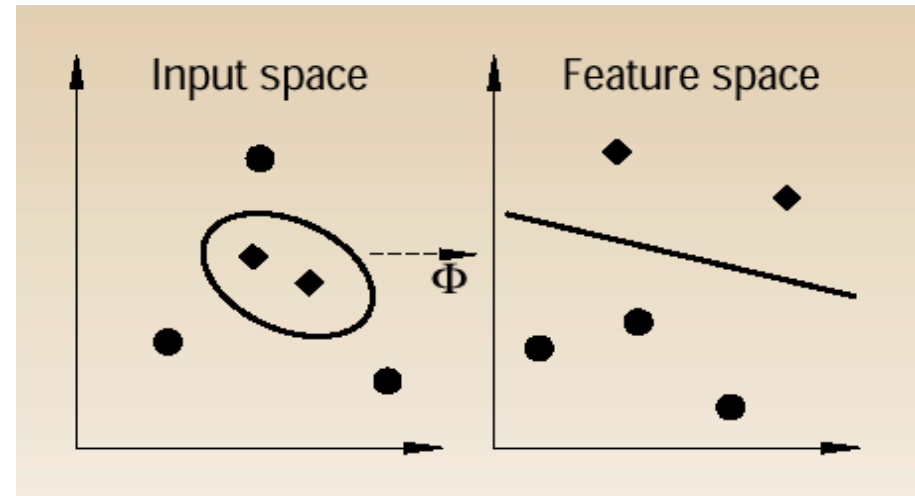
- Kernel

- La valutazione della funzione di decisione richiede il prodotto scalare ma mai i pattern rimappati in forma esplicita

$$\phi(x)$$

- Il prodotto scalare viene calcolato attraverso la funzione kernel

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



Funzione di classificazione: forma duale

$$\text{sgn}(\vec{w} \cdot \vec{x} + b) = \text{sgn}\left(\sum_{j=1..l} \alpha_j y_j \vec{x}_j \cdot \vec{x} + b \right)$$

- Notare che i dati appaiono solo nel prodotto scalare
- La matrice $G = \left(\langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle \right)_{i,j=1}^l$ è chiamata Gram matrix

Algoritmo duale del Perceptrone e le funzioni Kernel

- Possiamo riscrivere la funzione di decisione:

$$\begin{aligned} h(x) &= \text{sgn}(\vec{w} \cdot \phi(\vec{x}) + b) = \text{sgn}\left(\sum_{j=1..l} \alpha_j y_j \phi(\vec{x}_j) \cdot \phi(\vec{x}) + b\right) = \\ &= \text{sgn}\left(\sum_{i=1..l} \alpha_j y_j k(\vec{x}_j, \vec{x}) + b\right) \end{aligned}$$

- ... e nel processo di aggiornamento

$$y_i \left(\sum_{j=1..l} \alpha_j y_j \phi(\vec{x}_j) \right) \cdot \phi(\vec{x}_i) + b = \sum_{j=1..l} \alpha_j y_i y_j k(\vec{x}_j, \vec{x}_i) + b$$

Kernels in Support Vector Machines

- Nel problema Soft Margin SVMs si deve massimizzare:

$$\sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y_i y_j \alpha_i \alpha_j \vec{x}_i \cdot \vec{x}_j + \frac{1}{2C} \vec{a} \cdot \vec{a} - \frac{1}{C} \vec{a} \cdot \vec{a}$$

Usando le funzioni kernel possiamo riscrivere il problema come:

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

Definizione delle Funzioni Kernel

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.

- Le funzioni kernel esprimono mappature implicite di questo tipo

$$\vec{x} \in \mathfrak{R}^n, \quad \vec{\phi}(\vec{x}) = (\phi_1(\vec{x}), \phi_2(\vec{x}), \dots, \phi_m(\vec{x})) \in \mathfrak{R}^m$$

Validità delle funzioni kernel (1)

Def. B.11 *Eigen Values*

Given a matrix $\mathbf{A} \in \mathbb{R}^m \times \mathbb{R}^n$, an eigenvalue λ and an eigenvector $\vec{x} \in \mathbb{R}^n - \{\vec{0}\}$ 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 $i \neq j$ $i = 1, \dots, m$ and $j = 1, \dots, n$, 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).

Validità delle funzioni kernel (2)

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).

- L'idea principale di questa proposizione è che se la Gram matrix è semidefinita positiva allora esiste il mapping ϕ che realizza la funzione kernel, cioè uno spazio F in cui la separabilità è espressa in modo migliore

Feature Spaces and Kernels

- Esempio di Kernel
 - Polynomial kernel

- Se $d=2$ e $k(x, y) = (x \cdot y)^d$
 $x, y \in R^2$

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

Polynomial Kernel (n dimensions)

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

General Polynomial Kernel (n dimensions)

$$\begin{aligned}
 (\vec{x} \cdot \vec{z} + c)^2 &= \left(\sum_{i=1}^n x_i z_i + c \right)^2 = \left(\sum_{i=1}^n x_i z_i + c \right) \left(\sum_{j=1}^n x_j z_j + c \right) = \\
 &= \sum_{i=1}^n \sum_{j=1}^n x_i x_j z_i z_j + 2c \sum_{i=1}^n x_i z_i + c^2 = \\
 &= \sum_{i,j \in \{1, \dots, n\}} (x_i x_j) (z_i z_j) + \sum_{i=1}^n (\sqrt{2c} x_i) (\sqrt{2c} z_i) + c^2
 \end{aligned}$$

Polynomial kernel and the conjunction of features

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

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

- More expressive, as (x_1x_2) encodes *stock+market vs. downtown+market* features
- We can smartly compute the scalar product as

$$\begin{aligned} \Phi(\vec{x}) \cdot \Phi(\vec{z}) &= (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1) \cdot (z_1^2, z_2^2, \sqrt{2}z_1z_2, \sqrt{2}z_1, \sqrt{2}z_2, 1) = \\ &= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 + 2x_1z_1 + 2x_2z_2 + 1 = \\ &= (x_1z_1 + x_2z_2 + 1)^2 = \boxed{(\vec{x} \cdot \vec{z} + 1)^2} = K_{p_2}(\vec{x}, \vec{z}) \end{aligned}$$

Polynomial kernel



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

Architettura di una SVM

- Classificatore non lineare (basato su kernel)
- La funzione di decisione è

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

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

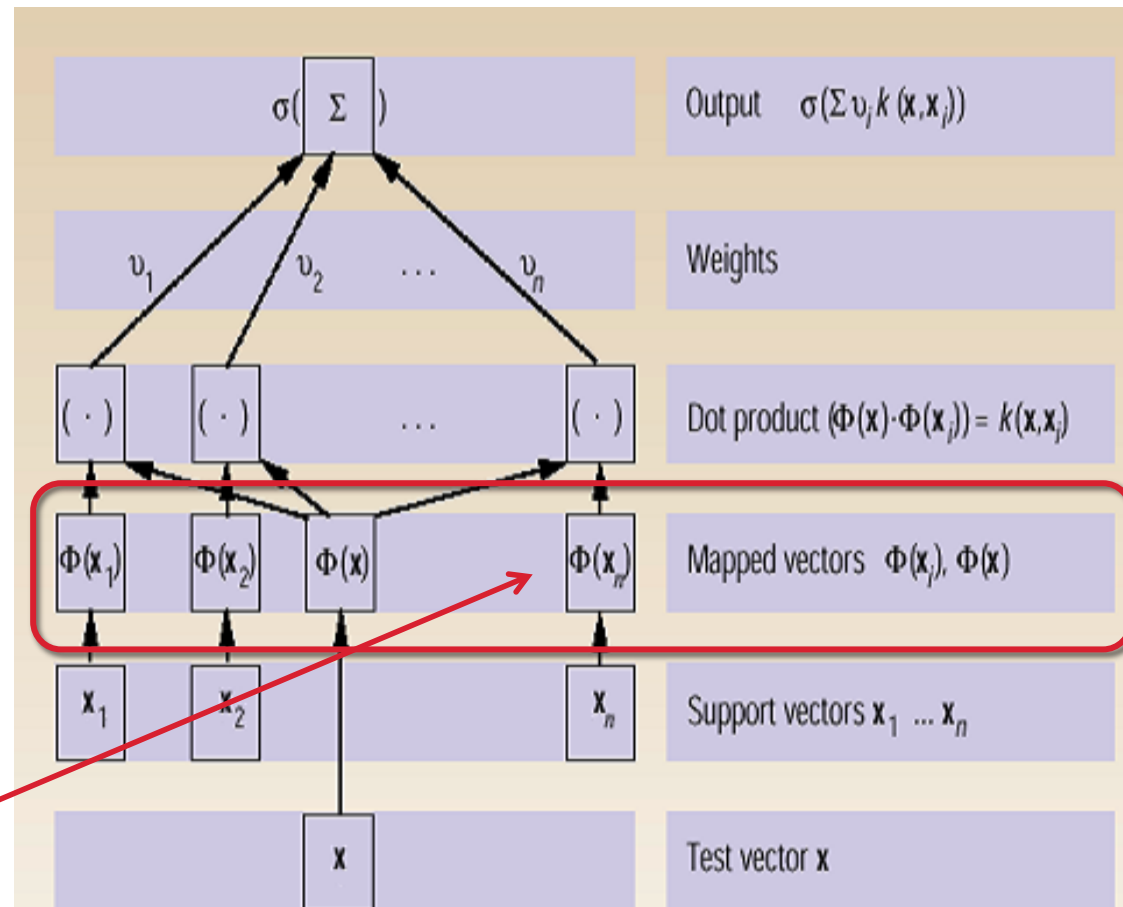
$\varphi(x_i)$ sostituisce ogni

esempio di training x_i

$$v_i = \alpha_i y_i$$

v_i sono calcolate attraverso la soluzione del problema di ottimizzazione

La funzione di mapping non è mai calcolata, ma implicita nella funzione kernel



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.

Formal Definition

Sottosequenza di indici ordinati
e non contigui di $(1, \dots, |s|)$

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

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

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

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

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

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*.



$\phi(\text{Bank}) = (\lambda, 0, \lambda, \lambda, \lambda, \lambda^2, \lambda^2, \lambda^3, 0, \lambda^4, 0, \lambda^2, \lambda^3, \lambda^3, \dots)$

$\phi(\text{Rank}) = (0, \lambda, \lambda, \lambda, \lambda, 0, 0, 0, \lambda^3, 0, \lambda^4, \lambda^2, \lambda^3, \lambda^3, \dots)$

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

▪Schriank and Rank

An example of string kernel computation

- $\phi_a(\text{Bank}) = \phi_a(\text{Rank}) = \lambda^{(i_1 - i_1 + 1)} = \lambda^{(2 - 2 + 1)} = \lambda,$
- $\phi_n(\text{Bank}) = \phi_n(\text{Rank}) = \lambda^{(i_1 - i_1 + 1)} = \lambda^{(3 - 3 + 1)} = \lambda,$
- $\phi_k(\text{Bank}) = \phi_k(\text{Rank}) = \lambda^{(i_1 - i_1 + 1)} = \lambda^{(4 - 4 + 1)} = \lambda,$
- $\phi_{an}(\text{Bank}) = \phi_{an}(\text{Rank}) = \lambda^{(i_1 - i_2 + 1)} = \lambda^{(3 - 2 + 1)} = \lambda^2,$
- $\phi_{ank}(\text{Bank}) = \phi_{ank}(\text{Rank}) = \lambda^{(i_1 - i_3 + 1)} = \lambda^{(4 - 2 + 1)} = \lambda^3,$
- $\phi_{nk}(\text{Bank}) = \phi_{nk}(\text{Rank}) = \lambda^{(i_1 - i_2 + 1)} = \lambda^{(4 - 3 + 1)} = \lambda^2,$
- $\phi_{ak}(\text{Bank}) = \phi_{ak}(\text{Rank}) = \lambda^{(i_1 - i_2 + 1)} = \lambda^{(4 - 2 + 1)} = \lambda^3.$

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

Tree Kernels

- Cio' che è stato mostrato per la gestione di sequenze (stringhe) nell'apprendimento di una SVM non dipende strettamente dalla nozione di sequenza, ma si applica a strutture piu' complesse
 - Alberi, che sono caratterizzati da un molteplicità sottosequenze, che corrispondono ai cammini radice-foglie
 - Grafi, che sono caratterizzati da piu' alberi, quindi da una maggiore molteplicità di sottosequenze

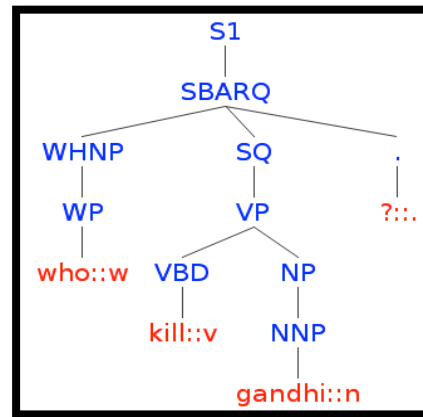
Tree kernels

- 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

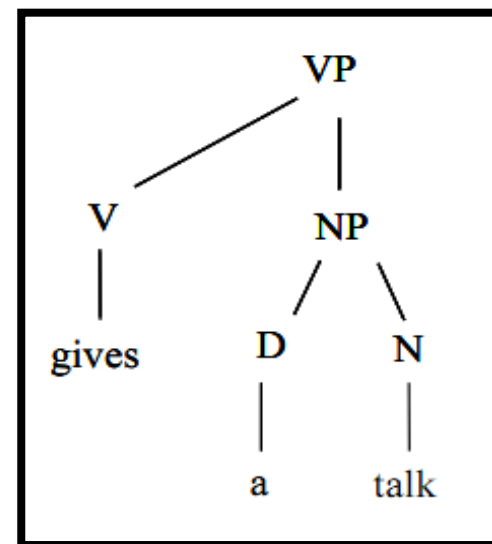


Tree kernels are natural way to exploit syntactic information from sentence parse trees

- 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 ***constituency*** tree



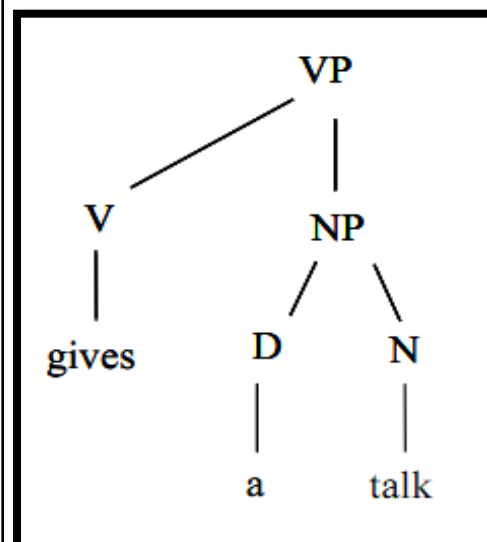
Syntactic Parsing and CFG

- Formal Definition: a context free grammar (CFG) is a 4-tuple $G=(N, \Sigma, R, S)$ where:
 - N is a set of non-terminal symbols
 - Σ is a set of terminal symbols
 - R is a set of production rules of the form $X \rightarrow Y_1Y_2 \cdots Y_n$
for $n \geq 0, X \in N, Y_i \in (N \cup \Sigma)$
 - $S \in N$ is a distinguished start symbol

Syntactic Parsing and CFG (2)

- $N = \{S, NP, VP, PP, DT, Vi, Vt, NN, IN\}$
- $S = S$
- $\Sigma = \{sleeps, saw, gives, man, woman, telescope, talk, with, in\}$

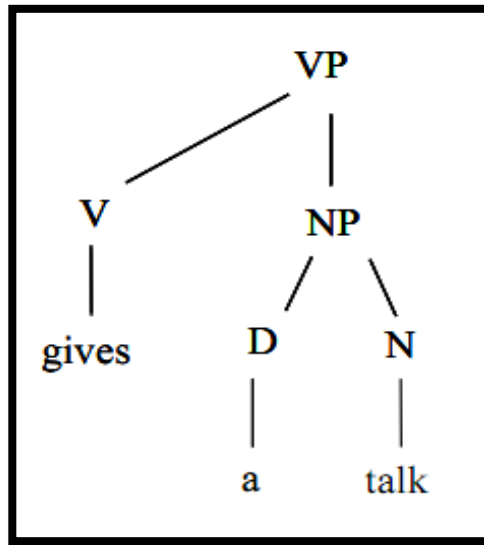
R=	S	→	NP VP	Vi	→	<i>sleeps</i>
	S	→	VP NP	Vt	→	<i>saw</i>
			...	Vt	→	<i>gives</i>
	VP	→	Vi	...		
	VP	→	Vt NP	NN	→	<i>man</i>
	VP	→	VP PP	NN	→	<i>woman</i>
			...	NN	→	<i>telescope</i>
	NP	→	DT NN	...		
	NP	→	NP PP	DT	→	<i>the</i>
			...	DT	→	<i>a</i>
	PP	→	IN NP	...		
			...	IN	→	<i>with</i>
			...	IN	→	<i>in</i>



Note

S=sentence
 VP=verb phrase
 NP=noun phrase
 PP=prepositional phrase
 DT=determiner
 Vi=intransitive verb
 Vt=transitive verb
 NN=noun
 IN=preposition

The Collins and Duffy's Tree Kernel

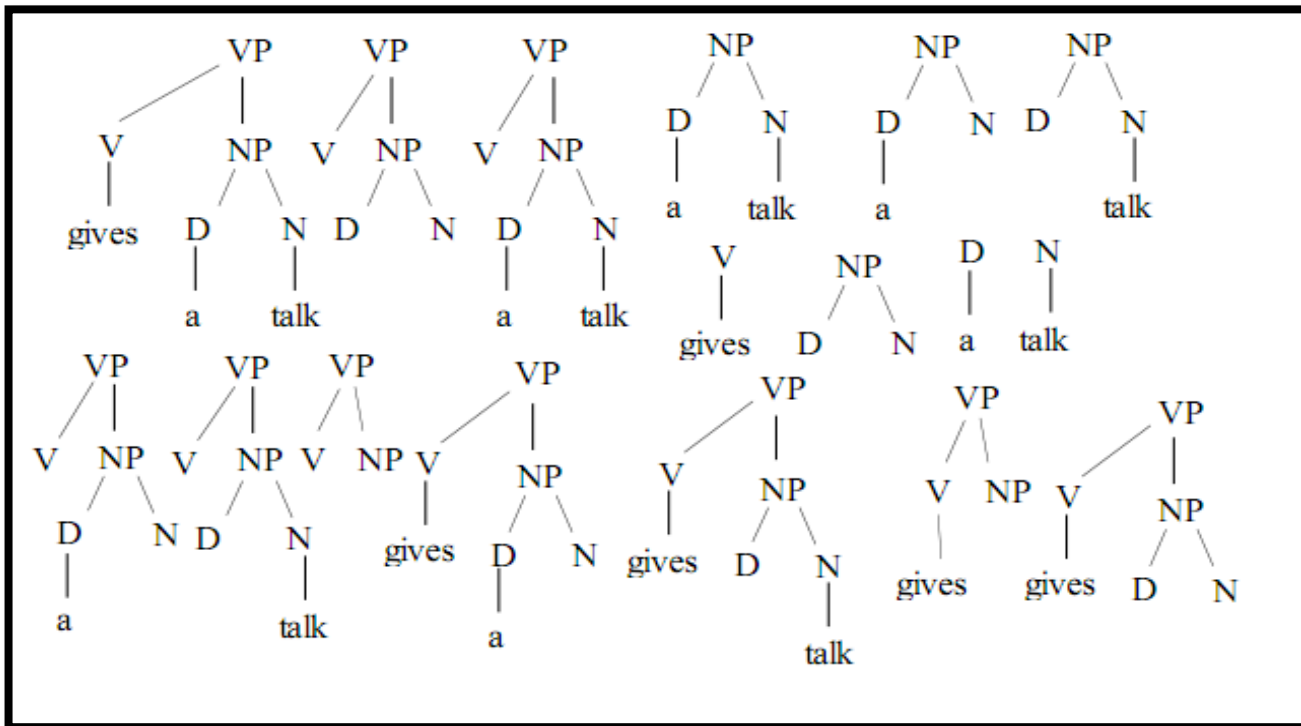


Given a constituency tree

The overall fragment set

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

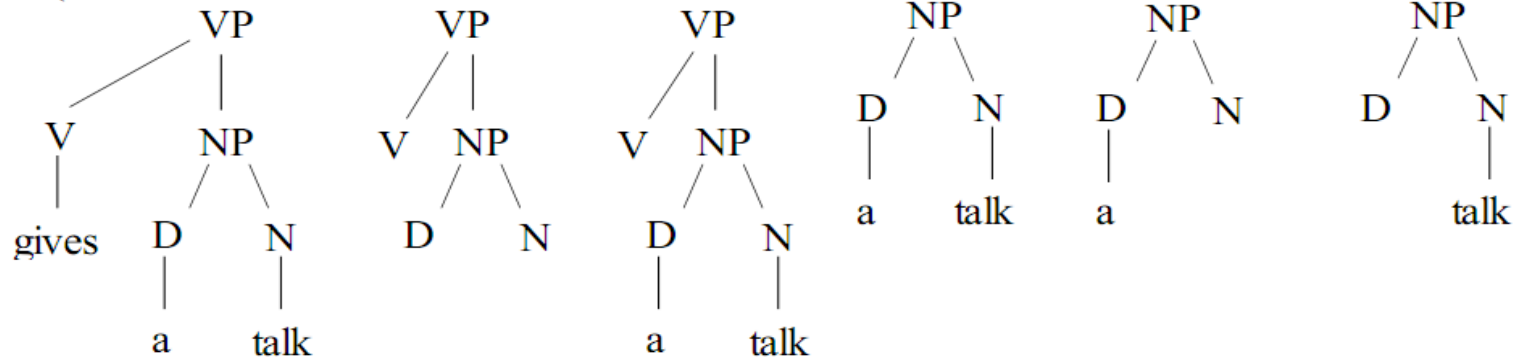
- For each node the production rules must be respected, i.e. we can remove “0 or all children at a time”
- It is also known as **Syntactic Tree Kernel**



Explicit feature space

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

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



$\vec{x}_1 \cdot \vec{x}_2$ 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

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

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

$\Delta(n_1, n_2) = 0$, **if the productions are different else**

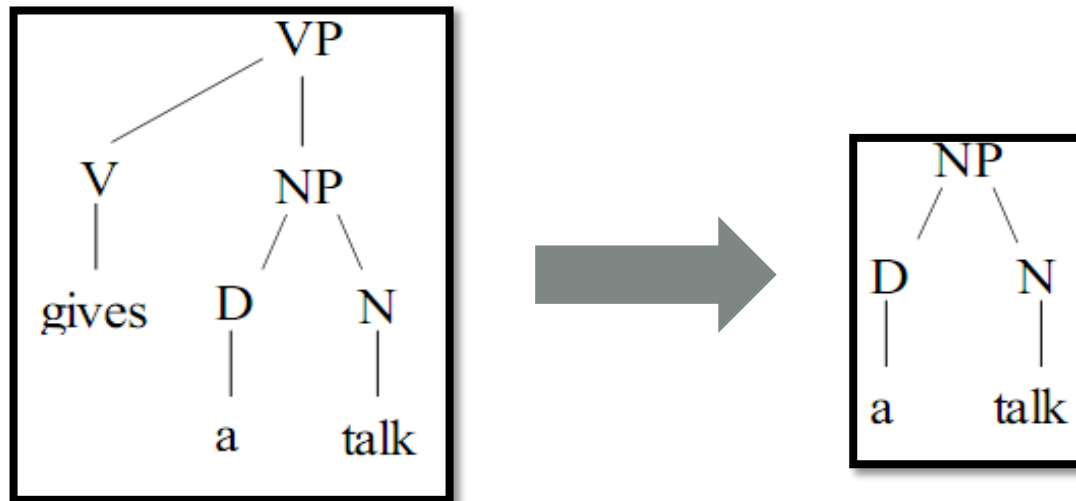
$\Delta(n_1, n_2) = 1$, **if pre - terminals else**

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

Weighting

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

$\Delta(n_1, n_2) = \lambda$, **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

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

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

- La forma duale del problema di ottimizzazione di una SVM dipende solo dal prodotto scalare degli esempi di addestramento e NON dalla loro esplicita rappresentazione vettoriale (come nel perceptrone)
- Questo suggerisce di sfruttare questa proprietà per
 - Definire funzioni capaci di calcolare in modo il prodotto scalare a partire dalla rappresentazione originale
 - Utilizzando quindi rappresentazioni (cioè spazi di feature) più complesse(i) in modo implicito
- Tale ricerca di spazi linearmente separabili
 - Mantiene le proprietà matematiche che garantiscono la minimizzazione dell'errore atteso
 - Contiene la complessità del processo di training e di classificazione

Summary (2)

- Affinchè una funzione $K(.,.)$ sia un kernel la gram matrix corrispondente deve essere semi-definita positiva
- Possono essere fornite anche semplici prove empiriche di tale proprietà sui data set di addestramento
- Sono state discusse:
 - Funzioni kernel di base (per esempio I kernel polinomiali di grado 2)
 - Funzioni kernel dipendenti dalla struttura del task
 - String (Sequence) kernels
 - Semantic kernel

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