Docenti: **Giorgio Valentini**
Matteo Re

UNIVERSITÀ DEGLI STUDI DI MILANO

Insegnamento: **Bioinformatica**
A.A. 2012-2013 semestre I

C.d.l. **Informatica**

Cheminformatica in R

Giorgio Valentini
e –mail: *valentini@dsi.unimi.it*
http://homes.dsi.unimi.it/~valenti

Matteo Re
e –mail: *re@dsi.unimi.it*
http://homes.dsi.unimi.it/~re

DSI – Dipartimento di Scienze dell' Informazione
Università degli Studi di Milano

Cheminformatica

- **Cheminformatica** è una disciplina definita recentemente (**1998**) che si pone come obiettivo quello di integrare informazioni disponibili in banche dati pubbliche e riguardanti farmaci, molecole e processi patologici in modo da produrre nuove conoscenze che possano essere di supporto durante il processo di **sviluppo di nuovi farmaci**.

Cheminformatica

In **R** esistono diverse librerie contenenti strumenti utilizzabili in esperimenti cheminformatici. In particolare i dati che utilizzeremo per i nostri test sono stati prodotti utilizzando le librerie R:

- rcdk (CRAN)
- ChemmineR (Bioconductor)

Cheminformatica

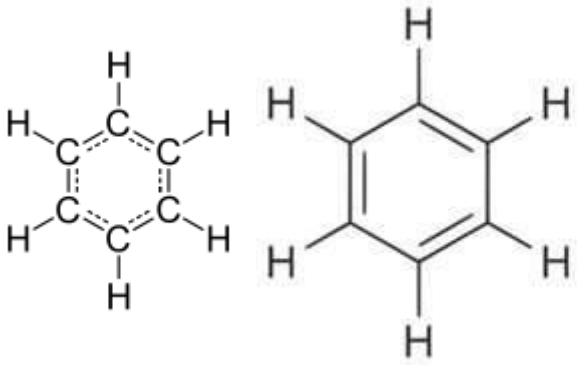
La realizzazione di una generica analisi cheminformatica pone alcuni problemi generali:

- **Rappresentazione** di molecole in modo che esse possano essere analizzate con un calcolatore.
- Definizione di nozioni di **similarità** tra molecole.
- Progettazione/implementazione di test che possano essere utilizzati per **predire un effetto della molecola su un sistema biologico o una caratteristica rilevante delle molecole in esame**

1 Cheminformatica : rappresentazione di molecole

Molecola: «gruppo elettricamente neutro di atomi tenuti insieme da legami chimici di tipo covalente»

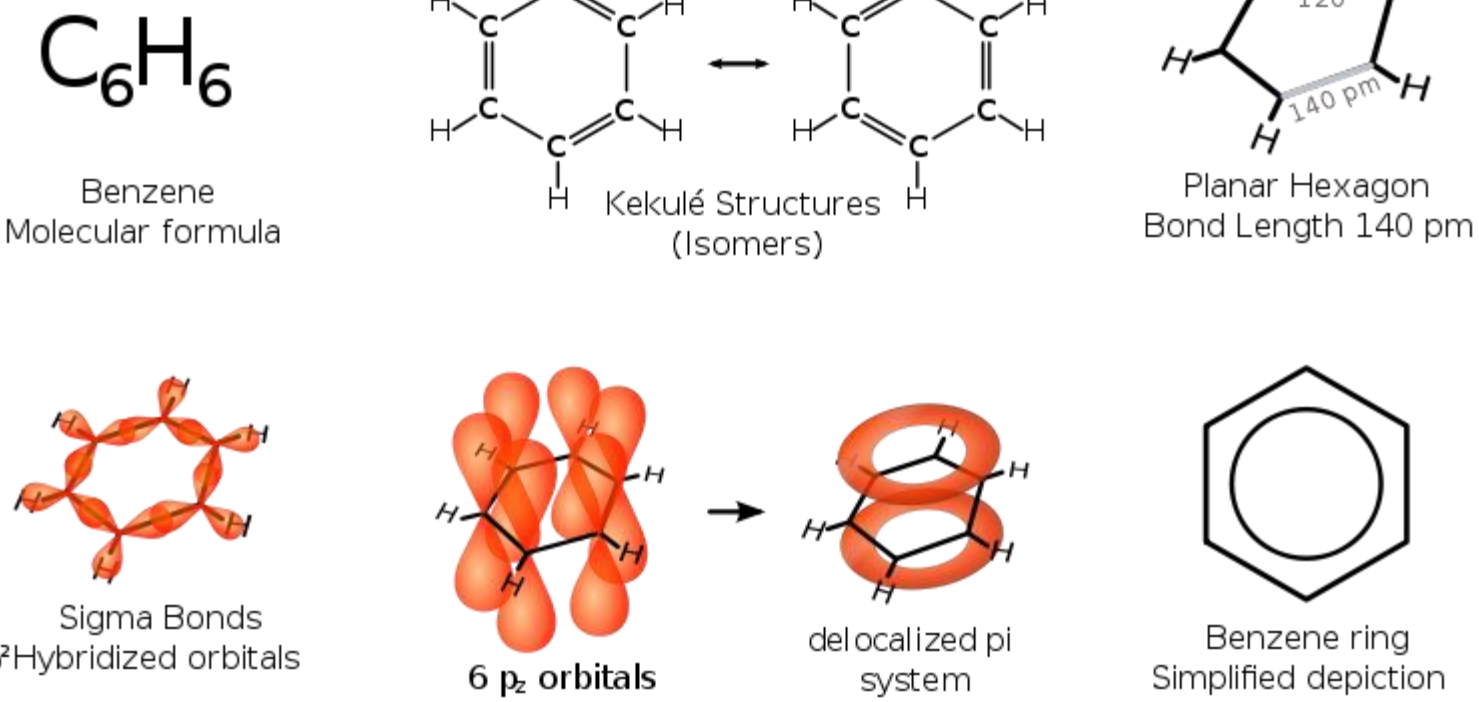
benzene:



C_6H_6

Sono tutte rappresentazioni valide ma <u>incomprensibili</u> per un computer

Cheminformatica : rappresentazione di molecole



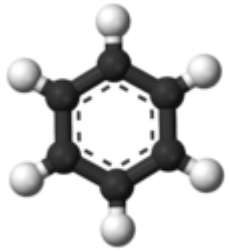
E’ un **problema di codifica**. Ci sono molti modi di rappresentare la struttura di questa molecola. Dobbiamo trovarne uno che sia adatto ad essere «compreso» da una macchina.

Cheminformatica : rappresentazione di molecole

Sono stati proposti diversi formati di file adatti a rappresentare la struttura delle molecole. Alcuni permettono di rappresentare solo la struttura, altri permettono l'inclusione di informazioni aggiuntive.

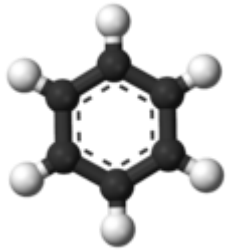
- MDL **MOL** (*.mol) : permette di codificare atomi, legami tra di essi, coordinate atomiche. Il file MOL (o molfile) contiene alcune righe di intestazione, la **Connection Table** (CT) contenente informazioni sugli atomi, una sezione dedicata ai **legami tra atomi** e sezioni aggiuntive adatte a contenere eventuali informazioni più complesse.
- Structure Data Format (**SDF**) è una estensione del formato MOL adatta a rappresentare informazioni aggiuntive più complesse e a gestire insiemi di molecole.
- Simplified Molecular Input Line Entry Specification (**SMILES**) rappresenta ogni molecola utilizzando una sola riga di testo

MOL file



```
1   benzene
2    ACD/Labs0812062058
3
4      6  6  0  0  0  0  0  0  0  0  1 V2000
5        1.9050   -0.7932    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
6        1.9050   -2.1232    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
7        0.7531   -0.1282    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
8        0.7531   -2.7882    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
9       -0.3987   -0.7932    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
10      -0.3987   -2.1232    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
11     2  1  1  0  0  0  0
12     3  1  2  0  0  0  0
13     4  2  2  0  0  0  0
14     5  3  1  0  0  0  0
15     6  4  1  0  0  0  0
16     6  5  2  0  0  0  0
17    M  END
18    $$$$
...
```

MOL file



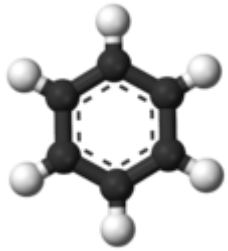
intestazione

Connection Table (CT)

```
1   benzene
2    ACD/Labs0812062058
3
4     6  6  0  0  0  0  0  0  0  0  1 V2000
5       1.9050   -0.7932    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
6       1.9050   -2.1232    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
7       0.7531   -0.1282    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
8       0.7531   -2.7882    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
9      -0.3987   -0.7932    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
10     -0.3987   -2.1232    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
11    2  1  1  0  0  0  0
12    3  1  2  0  0  0  0
13    4  2  2  0  0  0  0
14    5  3  1  0  0  0  0
15    6  4  1  0  0  0  0
16    6  5  2  0  0  0  0
17   M  END
18   $$$$
...
```

Conteggio: 6 atomi, 6 legami, … , standard V2000

MOL file



intestazione

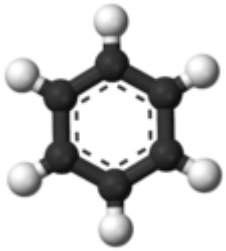
Connection Table (CT)

```
1   benzene
2    ACD/Labs0812062058
3
4      6  6  0  0  0  0  0  0  0  0  1 V2000
5       1.9050   -0.7932    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
6       1.9050   -2.1232    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
7       0.7531   -0.1282    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
8       0.7531   -2.7882    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
9      -0.3987   -0.7932    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
10     -0.3987   -2.1232    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
11     2  1  1  0  0  0  0
12     3  1  2  0  0  0  0
13     4  2  2  0  0  0  0
14     5  3  1  0  0  0  0
15     6  4  1  0  0  0  0
16     6  5  2  0  0  0  0
17    M  END
18    $$$$
...
```

Atomi e coordinate

Tipo di atomo

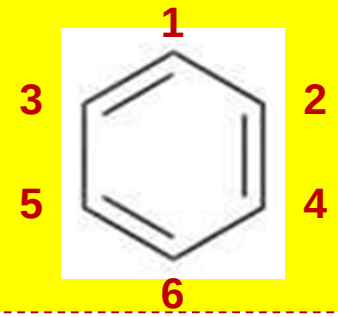
MOL file



intestazione

Connection Table (CT)

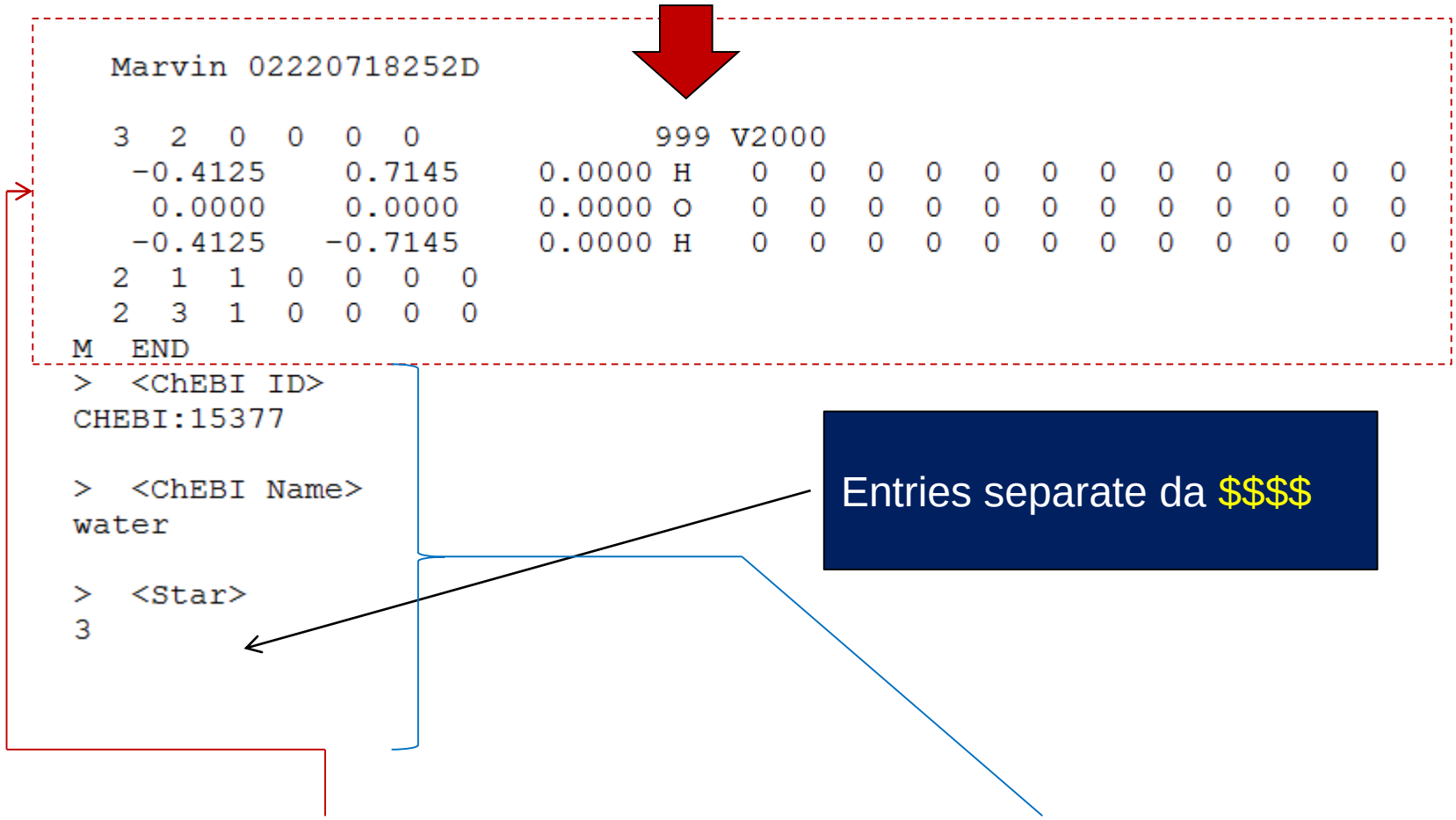
```
1   benzene
2    ACD/Labs0812062058
3
4     6  6  0  0  0  0  0  0  0  0  1 V2000
5       1.9050   -0.7932    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
6       1.9050   -2.1232    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
7       0.7531   -0.1282    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
8       0.7531   -2.7882    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
9      -0.3987   -0.7932    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
10     -0.3987   -2.1232    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
11    2  1  1  0  0  0  0
12    3  1  2  0  0  0  0
13    4  2  2  0  0  0  0
14    5  3  1  0  0  0  0
15    6  4  1  0  0  0  0
16    6  5  2  0  0  0  0
17  M  END
18   $$$$
...
```



Definizione **legami** (da, a, tipo,…)

SDF file

Che molecola è ?



SDF: **formato MOL** arricchito con **informazioni aggiuntive**

SMILES file

Simplified Molecular Input Line Entry Specification (**SMILES**) è un sistema di codifica per la struttura delle molecole in grado di convertire una struttura chimica **in una stringa di testo** seguendo un set di regole predefinite. le stringhe SMILES contengono tipi di atomi e legami tra di essi ma non contengono coordinate 2D o 3D.

Gli atomi H non sono rappresentati. Altri atomi vengono rappresentati mediante il loro simbolo chimico, ad es. B, C, N, O, F, P, S, Cl, Br, e I. Il simbolo "=" sappresenta il doppio legame ed il simbolo "#" rappresenta il triplo legame. Gruppi di atomi (es, CH3 per il metile) vengono racchiusi tra parentesi. I cicli sono espressi da coppie di numeri (ad es. la rappresentazione SMILES dell'anello del benzene inizia e finisce con **1**: C**1** ... **1** .

SMILES file

Esempi :

Nome	Formula	stringa SMILES
Metano	CH4	C
Acqua	H2O	O
Etanolo	C2H6O	CCO
Benzene	C6H6	C**1**=CC=CC=C**1** oppure c**1**ccccc**1**
Etilene	C2H4	C=C

Idrogeni <u>non</u> sono rappresentati

anello

SMILES file

```
DB00116     NC1=NC(=O)C2=C(NCC(CNC3=CC=C(C=C3)C(=O)N[C@@H](CCC(O)=O)C(O)=O)N2)N1
DB00117     N[C@@H](CC1=CN=CN1)C(O)=O
DB00118     C[S+](CC[C@H](N)C(O)=O)C[C@H]1O[C@H]([C@H](O)[C@@H]1O)N1C=NC2=C1N=CN=C2N
DB00119     CC(=O)C(O)=O
DB00120     N[C@@H](CC1=CC=CC=C1)C(O)=O
DB00121     [H][C@]12CS[C@@H](CCCCC(O)=O)[C@@]1([H])NC(=O)N2
DB00122     C[N+](C)(C)CCO
DB00123     NCCCC[C@H](N)C(O)=O
DB00125     N[C@@H](CCCNC(N)=N)C(O)=O
DB00126     [H][C@@]1(OC(=O)C(O)=C1O)[C@@H](O)CO
DB00127     NCCCNCCCCNCCCN
DB00128     N[C@@H](CC(O)=O)C(O)=O
DB00129     NCCC[C@H](N)C(O)=O
DB00130     N[C@@H](CCC(N)=O)C(O)=O
DB00131     NC1=NC=NC2=C1N=CN2[C@@H]1O[C@H](COP(O)(O)=O)[C@@H](O)[C@H]1O
DB00132     CC\C=C/C\C=C/C\C=C/CCCCCCCC(O)=O
DB00133     N[C@@H](CO)C(O)=O
DB00134     CSCC[C@H](N)C(O)=O
DB00135     N[C@@H](CC1=CC=C(O)C=C1)C(O)=O
DB00136     [H][C@@]1(CC[C@@]2([H])C(CCC[C@]12C)=CC=C1C[C@@H](O)C[C@H](O)C1=C)[C@H](C)CCCC(C)(C)O
DB00137     CC(\C=C\C=C(C)\C=C\C1C(C)=CC(O)CC1(C)C)=C/C=C/C=C(C)/C=C/C=C(C)/C=C/C1=C(C)CC(O)CC1(C)C
DB00138     N[C@@H](CSSC[C@H](N)C(O)=O)C(O)=O
DB00139     OC(=O)CCC(O)=O
DB00140     CC1=CC2=C(C=C1C)N(C[C@H](O)[C@H](O)[C@H](O)CO)C1=NC(=O)NC(=O)C1=N2
DB00141     CC(=O)N[C@H]1C(O)O[C@H](CO)[C@@H](O)[C@@H]1O
DB00142     N[C@@H](CCC(O)=O)C(O)=O
DB00143     N[C@@H](CCC(=O)N[C@@H](CS)C(=O)NCC(O)=O)C(O)=O
DB00144     CCCC(=O)O[C@H](COC(=O)CC)COP(O)(=O)OC[C@H](N)C(O)=O
DB00145     NCC(O)=O
...
```

ID molecola
(Drugbank ID)

stringa SMILES

Cheminformatica : librerie R

Grazie all'utilizzo di apposite librerie open source R è possibile scaricare collezioni di **descrittori molecolari** (ad es. in formato SMILES) da banche dati pubbliche (una delle più utilizzate è DrugBank) e utilizzare queste informazioni per creare matrici di similarità tra molecole. Due package R che possono essere utilizzati per svolgere queste operazioni sono **rcdk e ChemmineR**.

Installazione:

rcdk :

```
> install.packages('rcdk', dependencies=TRUE)
```

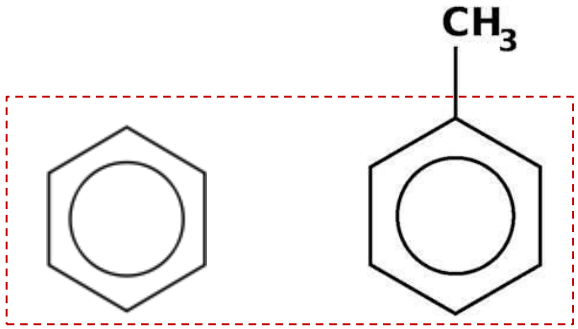
ChemmineR :

```
> source("http://bioconductor.org/biocLite.R")

> biocLite("ChemmineR")
```

2 Cheminformatica : confronto tra molecole

Per poter realizzare un test basato sull'utilizzo di un random walk su grafo dobbiamo riuscire calcolare in modo automatico quanto sono «simili» due molecole. Come possiamo riuscirci?

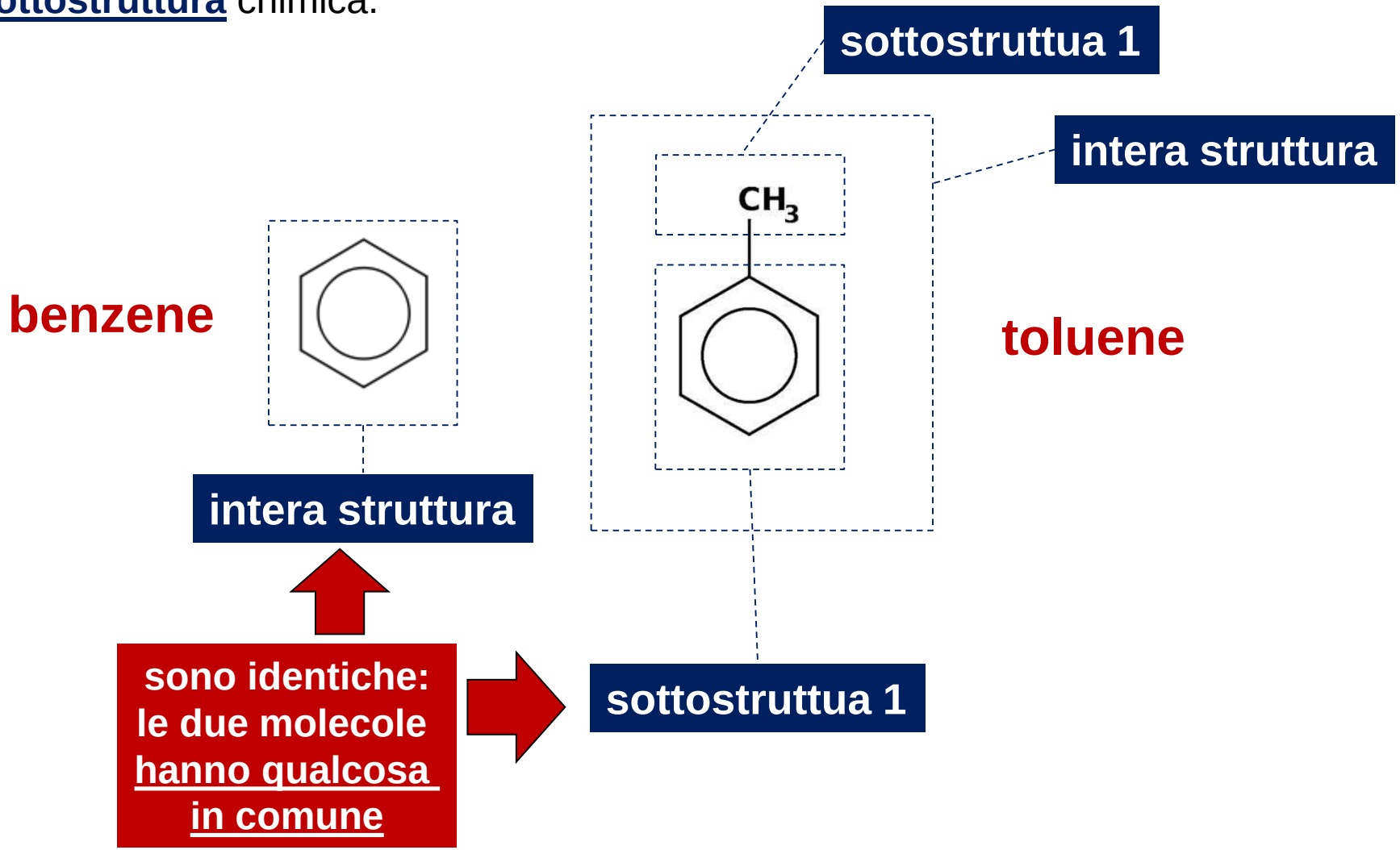


benzene toluene

«A occhio» le loro strutture sono simili … ma «a occhio» per un calcolatore non ha senso

Il concetto di fingerprint molecolare

Le fingerprint molecolari sono strettamente correlate ai concetti di **struttura** e **sottostruttura** chimica.

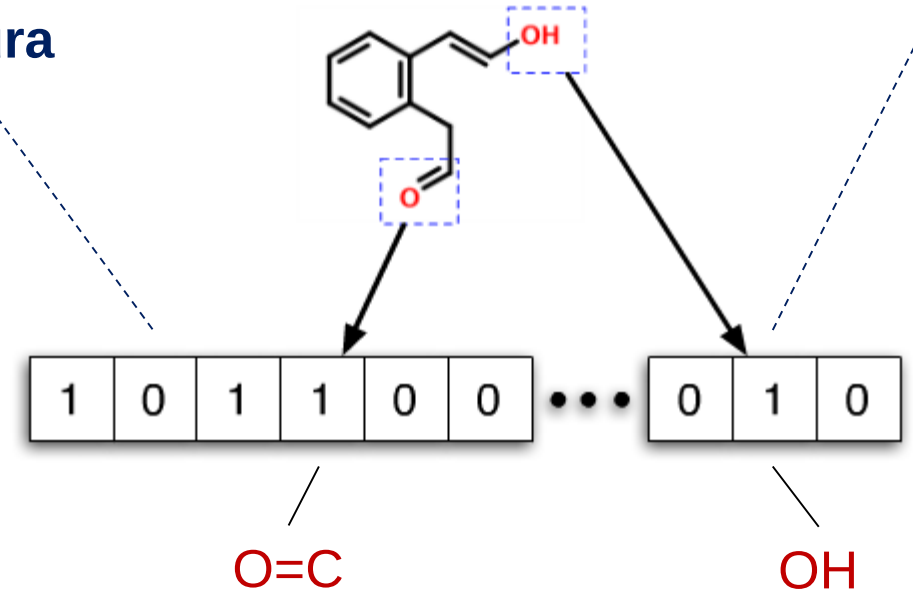


Il concetto di fingerprint molecolare

Le fingerprint molecolari sono strettamente correlate ai concetti di **struttura** e **sottostruttura** chimica.

0 : sottostruttura assente nella molecola

1 : sottostruttura presente nella molecola



Codifica **vettoriale** (vettore valori logici) : comprensibile per il calcolatore

Confronto di molecole mediante fingerprint

Una volta codificata la struttura delle molecole mediante fingerprint il loro confronto è relativamente semplice. L'obiettivo è quello di capire quante sottostrutture sono condivise da due molecole ed utilizzare questa informazione per calcolare uno score di similarità.

Date die fingerprint I e II che codificano le sottostrutture di due molecole che vogliamo confrontare definiamo:

a : numero di sottostrutture presente in fingerprint I **MA NON** in fingerprint II
b : numero di sottostrutture pesente in fingerprint I **E** in fingerprint II
c : numero di sottostrutture presente in fingerprint II **MA NON** in fingerprint I

Coefficiente di TANIMOTO :

$$\frac{b}{a + b + c}$$

Numero compreso tra 0 e 1 . 1 indica identità strutturale, 0 indica diversità totale delle strutture.

Confronto di molecole mediante fingerprint

Una volta codificata la struttura delle molecole mediante fingerprint il loro confronto è relativamente semplice. L'obiettivo è quello di capire quante sottostrutture sono condivise da due molecole ed utilizzare questa informazione per calcolare uno score di similarità.

Date die fingerprint I e II che codificano le sottostrutture di due molecole che vogliamo confrontare definiamo:

a : numero di sottostrutture presente in fingerprint I **MA NON** in fingerprint II
b : numero di sottostrutture pesente in fingerprint I **E** in fingerprint II
c : numero di sottostrutture presente in fingerprint II **MA NON** in fingerprint I

Coefficiente di TANIMOTO :

$$\frac{b}{a + b + c}$$

Numero compreso tra 0 e 1 . 1 indica identità strutturale, 0 indica diversità totale delle strutture.

Cheminformatica : test di riposizionamento di farmaci

Grazie alle librerie **rcdk e ChemmineR** e ai dati scaricati da **DrugBank** (http://www.drugbank.ca/) è stata creata una matrice 1253 x 1253 contenente tutte le similarità pairwise (espresse tramite **coefficiente di Tanimoto**) tra tutte le molecole approvate per l'utilizzo come farmaci da FDA.

E' inoltre stata creata una matrice binaria di dimensione 1253 x 51 in cui le righe rappresentano i farmaci e le colonne 51 categorie terapeutiche definite da DrugBank. Se trovate un valore **1** nella cella all'intersezione tra l'i-esima riga e la j-esima colonna questo indica che il farmaco **i** appartiene alla categoria terapeutica **j** . Troverete uno **0** in caso contrario.

Le due matrici sono state salvate in un file binario compresso disponibile a questo indirizzo:

http://homes.di.unimi.it/~re/Corsi/Bioinfo12_DI/Bioinfo_DI_data_LezP1.rda

Cheminformatica : test di riposizionamento di farmaci

Le due matrici sono state salvate in un file binario compresso disponibile a questo indirizzo:

http://homes.di.unimi.it/~re/Corsi/Bioinfo12_DI/Bioinfo_DI_data_LezP1.rda

Il file contiene 2 variabili R (entrambe matrici) di nome:

tanimoto.bin.mat.**0.5**	matrice di adiacenza del grafo (matr. binaria)
DB3.labels	matrice delle etichette (mat. binaria)

Lo **0.5** nel nome della matrice di adiacenza indica il valore soglia utilizzato per la binarizzazione della matrice. Essa era una matrice composta da reali compresi tra 0 e 1 (inclusi) corrispondenti alle similarità pairwise (coeff. Tanimoto) tra coppie di molecole. Per trasformarla in una matrice binaria tutti gli elementi della matrice sono stati impostati a 1 se maggiori o uguali ad una soglia **s** e a 0 in caso contrario. La soglia utilizzata (presa dalla letteratura) è : **s = 0.5**

Cheminformatica : test di riposizionamento di farmaci

In questa esercitazione pratica ci proponiamo di:

•Caricare il file delle matrici in R
•Verificare la presenza delle matrici, e le loro dimensioni
•Estrarre un vettore binario corrispondente alla classe terapeutica «Penicillins»
•Scrivere una generica funzione R
•Scrivere una funzione R che avendo in input la matrice di adiacenza del grafo ed un vettore binario contenente le etichette della classe «Penicillins» esegua un random walk su grafo con punti di origine dei cammini equiprobabili e corrispondenti ai nodi (farmaci) della classe considerata. Il risultato ottenuto (vettore probabilità) verrà utilizzato per definire un **ordinamento dei nodi** che riflette al loro probabilità di appartenenza alla classe considerata.
•Valutazione delle performance di ranking

3 Linguaggio R : esecuzione dei programmi

- I programmi (sequenze di espressioni) possono essere eseguiti :
  - *Interattivamente*: ogni istruzione viene eseguita direttamente al prompt dei comandi
  - *Non interattivamente*: le espressioni sono lette da un file (tramite la funzione ***source***) ed eseguite dall' interprete una ad una in sequenza.
- Usare un text editor (ad esempio *Notepad++*, scaricabile gratuitamente dalla rete)

Esempio di script R

```
# Functional classification of yeast genes using gene expression data
library(yeastCC);
library(e1071);
data(spYCCES);
source("yeastGO.R");

# data preparation
Yeast.specific.TAS <- Get.yeast.GO.specific.classes(evidence="TAS");
Yeast.general.TAS <- Get.yeast.GO.all.classes(Yeast.specific.TAS);
load("Yeast.general.classes.TAS.object");
l <- Count.examples.per.class(Yeast.general.classes.TAS);
cl1.genes <- Extract.class(Yeast.general.classes.TAS,"GO:0000902"); cl2.genes <- Extract.class(Yeast.general.classes.TAS,"GO:0006092");
paste("GO:0000902", ":", get.Term.Definition("GO:0000902")); paste("GO:0006092", ":", get.Term.Definition("GO:0006092"));
exprs.cl1 <- exprs(spYCCES)[as.character(cl1.genes$gene.names),]; exprs.cl2 <- exprs(spYCCES)[as.character(cl2.genes$gene.names),];
exprs.cl1[is.na(exprs.cl1)] <- 0; exprs.cl2[is.na(exprs.cl2)] <- 0;

# classification of yeast genes
n1.test<-round(nrow(exprs.cl1)/3); n1.train<-nrow(exprs.cl1)-n1.test;
n2.test<-round(nrow(exprs.cl2)/3); n2.train<-nrow(exprs.cl2)-n2.test;
Xtrain<-rbind(exprs.cl1[1:n1.train,],exprs.cl2[1:n2.train,]);
Xtest<-rbind(exprs.cl1[(n1.train+1):nrow(exprs.cl1),],exprs.cl2[(n2.train+1):nrow(exprs.cl2),]);
ytrain<-as.factor(c(rep(1,n1.train),rep(2,n2.train)));
ytest<-as.factor(c(rep(1,n1.test),rep(2,n2.test)));
model <- svm(as.matrix(Xtrain),ytrain,type="C-classification", kernel="linear", cost=1, gamma=1, degree=2, coef0=1);
prediction <- predict(model,Xtest);
conf.matrix <- table(prediction,ytest);
sensitivity <- conf.matrix[1,1]/(conf.matrix[1,1]+conf.matrix[2,1]);
specificity <- conf.matrix[2,2]/(conf.matrix[2,2]+conf.matrix[1,2]);
accuracy <- (conf.matrix[1,1] + conf.matrix[2,2])/length(ytest);
```

Funzioni R definite dall'utente

- Abbiamo già visto esempi di funzioni disponibili in R
- Le funzioni in R possono anche essere definite dagli utenti
- I programmi in R possono essere realizzati tramite funzioni

Funzioni: sintassi

La sintassi per scrivere una funzione è:

`function (argomenti)  corpo_della_funzione`

- `function` è una parola chiave di R
- `Argomenti` è una lista eventualmente vuota di *argomenti formali* separati da virgole:
  `(arg1, arg2, ..., argN)`
- Un *argomento formale* può essere un simbolo o un'istruzione del tipo '`simbolo=espressione`'
- Il `corpo` può essere qualsiasi espressione valida in R. Spesso è costituito da un gruppo di espressioni **racchiuso fra parentesi graffe**

Funzioni: esempi (1)

Funzione per il calcolo della statistica di Golub
x,y : **vettori** di cui si vuole calcolare la statistica di golub
La funzione ritorna il valore della statistica di Golub

nome

argomenti

```
golub <- function(x,y){
  mx <- mean(x);
  my <- mean(y);
  vx <- sd(x);
  vy <- sd(y);
  g <- (mx-my)/(vx+vy);
  return(g);
}
```

La sequenza di istruzioni del corpo della funzione deve essere racchiusa fra parentesi quadre

Funzioni: esempi (2)

Utilizzo della funzione di Golub:

- La funzione golub è memorizzata nel file “golub.R” (ma potrebbe essere memorizzata in un file con nome diverso)
- Caricamento in memoria della funzione. Due possibilità:

1. `> source("golub.R")`
2. Dal menu File/Source R code ...

- Chiamata della funzione:

```
> x<-runif(5) # primo argomento della funzione
> x
[1] 0.6826218 0.9587295 0.4718516 0.8284525 0.2080131
> y<-runif(5) # secondo argomento della funzione
> y
[1] 0.6966353 0.0964740 0.4310154 0.1467449 0.2801970
> golub(x,y) # chiamata della funzione
[1] 0.5553528
```

Argomenti formali e attuali

x e y sono *argomenti formali*:

```
> golub <- function(x,y)  { ... }
```

Tali valori vengono sostituiti dagli ***argomenti attuali*** quando la funzione è chiamata:

```
> d1 <- runif(5)
> d2 <- runif(5)
```

`d1` e `d2` sono gli **argomenti attuali** che sostituiscono i formali e vengono effettivamente utilizzati all' interno della funzione:

```
> golub(d1,d2)
[1] 0.2218095
> d3 <- 1:5
> golub(d1,d3)
[1] -1.325527
```

Gli argomenti sono passati per valore

Le modifiche agli argomenti effettuate nel corpo delle funzioni non hanno effetto all' esterno delle funzioni stesse:

```
> fun1 <- function(x) { x <- x*2 }
> y<-4
> fun1(y)
> y
[1] 4
```

In altre parole i valori degli argomenti attuali sono modificabili all' interno della funzione stessa, ma non hanno alcun effetto sulla variabile dell' ambiente chiamante.

Nell' esempio precedente la copia di `x` locale alla funzione viene modificata, ma non viene modificato il valore della variabile `y` passata come argomento attuale alla funzione `fun1`

Modalità di assegnamento degli argomenti: assegnamento posizionale

Tramite questa modalità gli argomenti sono assegnati in base alla loro posizione nella lista degli argomenti:

```
> fun1 <- function (x, y, z, w) {}
> fun1(1,2,3,4)
```

L' argomento attuale 1 viene assegnato a *x*, 2 a *y*, 3 a *z* e 4 a *w*.

Altro esempio:

```
>  sub <- function (x, y) {x-y}
> sub(3,2) # x<-3 e y<-2
[1] 1
> sub(2,3) # x<-2 e y<-3
[1] -1
```

Modalità di assegnamento degli argomenti: assegnamento per nome

Tramite questa modalità gli argomenti sono assegnati in base al loro nome nella lista degli argomenti:

```
> fun1 <- function (x, y, z, w) {}
> fun1(x=1, y=2, z=3, w=4)
```

L' argomento attuale 1 viene assegnato a *x*, 2 a *y*, 3 a *z* e 4 a *w*.

Quando gli argomenti sono assegnati per nome **non è necessario rispettare l' ordine degli argomenti**:

```
fun1(y=2, w=4, z=3, x=1) ≡ fun1(x=1, y=2, z=3, w=4)
```

Ad esempio:

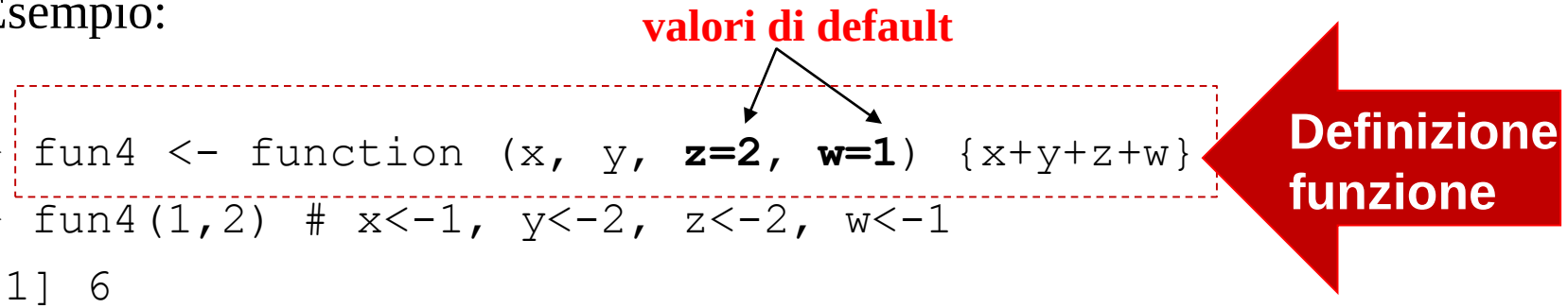
```
>  sub <- function (x, y) {x-y}
> sub(x=3,y=2) # x<-3 e y<-2
[1] 1
> sub(y=2,x=3) # x<-3 e y<-2
[1] 1
```

Modalità più sicura.
Da usare preferenzialmente rispetto all'assegnamento posizionale.

Valori di default per gli argomenti

E' possibile stabilire valori predefiniti per tutti o per parte degli argomenti: tali valori vengono assunti dalle variabili a meno che non vengano esplicitamente modificati nella chiamata della funzione.

Esempio:




```
> fun4 <- function (x, y, z=2, w=1) {x+y+z+w}
> fun4(1,2) # x<-1, y<-2, z<-2, w<-1
[1] 6
> fun4(1,2,5) # x<-1, y<-2, z<-5, w<-1
[1] 9
> fun4(1) # y non ha valore di default !
Error in fun4(1) : Argument "y" is missing, with no
   default
```

Algoritmo random walk su grafo

Librerie R necessarie: (comandi per l'installazione)

source("http://bioconductor.org/biocLite.R");
biocLite("limma");
biocLite("graph");
biocLite("RBGL");

Install.packages();

Nella finestra che compare scegliete un mirror, poi installate Le librerie **NetPreProc** e **PerfMeas**

Algoritmo random walk su grafo

Input:
- the adjacency matrix ***W*** of a graph G=<*V,E*>
- A subset of nodes *VC* having property *C*

I • Initialization of nodes:
if $v \in V_C$ then $p0(v) = 1 / |VC|$ else $p_0(v)=0$

II • Set transition matrix: $\boldsymbol{Q} = \boldsymbol{D}^{-1}\boldsymbol{W}$
where ***D*** is a diagonal matrix with
$d_{ii} = \Sigma_j \ w_{ij}$

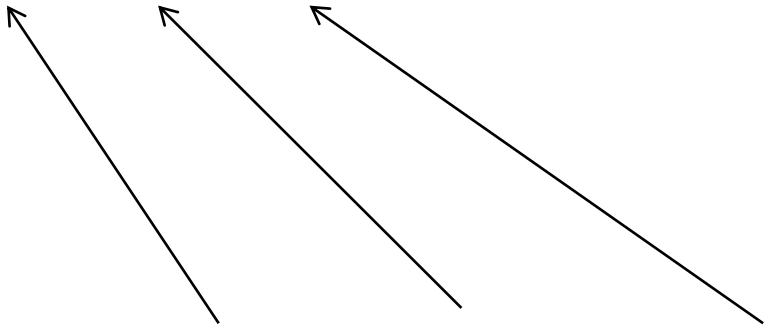
III • Iteratively update until convergence or until $t=k$
$\boldsymbol{p}_t = \boldsymbol{Q}^T \boldsymbol{p}_{t-1}$

IV Output: $\boldsymbol{p}_t$

Esempio di funzione : norm1

```
# definizione funzione norm1 :

norm1 <- function(x) {
 return(sum(abs(x)));
}
```



Questa funzione **ritorna** la **somma** dei **valori assoluti** dei componenti di un **vettore** passato in input.

Random walk: step I

```
# Inizializzazione vettore prob. Nodi

load("Bioinfo_DI_data_LezP1.rda"");

# n. drugs
n <- nrow(tanimoto.bin.mat.0.5);

# ind. positives
ind.pos <- which(DB3.labels[,47]==1);

# Prob. vectors
p0 <- numeric(n);
names(p0) <- rownames(tanimoto.bin.mat.0.5);
p <- numeric(n);
names(p) <- rownames(tanimoto.bin.mat.0.5);

# n. positives
n.positives <- length(ind.pos);

# Init. probs definition
p0[ind.pos] <- 1/n.positives;
```

Random walk: step II

```
# Set transition matrix

library(NetPreProc);

# M = D^-1 * W
M <- Prob.norm(tanimoto.bin.mat.0.5);
```

Per realizzare questa parte del test utilizziamo la funzione Prob.norm resa disponibile dalla libreria R *NetPreProc*.

Random walk: step III

```
# Iteratively update until convergence or
until t=k

tmax=1000;
eps=1e-10;

M <- t(M);
p <- p0;

 for (t in 1:tmax){
   pold <- p;
   p <- M %*% pold;
   if (norm1(p-pold) < eps){
    break();
   }
 }
```

Criteri di terminazione

Iterazione

$$p_t = Q^T p_{t-1}$$

Random walk: step IV

```
# Return p_t

return(p);


# modo più efficace: ritornare lista di
# variabili (ognuna associata ad un nome):

results <- list(p=p, ind.positives=ind.pos,
n.iter=t);

return(results);
```

Area sotto la curva ROC (AUC)

Ottenuti gli score associati alla probabilità di ogni nodo mediante random walk su grafo dobbiamo quantificare le performance di ranking.

Per ottenere questo risultato utilizzeremo una tecnica comunemente adottata in letteratura per quantificare le performance di algoritmi di classificazione/ranking : calcoleremo il valore dell'*area sotto la curva ROC, o* ***AUC***.

Per riuscirci utilizzeremo una libreria R appositamente sviluppata per fornire all'utente diverse funzioni in grado di valutare le performance in esperimenti di ranking o classificazione. Il nome della libreria è **PerfMeas**.

Useremo la funzione AUC.single resa disponibile dalla libreria R *PerfMeas*.

```
Library(PerfMeas);
```

Area sotto la curva ROC (AUC)

Il processo di ranking produce uno score per ogni nodo del grafo.

Questi score possono essere ordinati. Supponiamo di definire una soglia C e di far assumere a quest'ultima una serie di valori.

Ricordiamoci che gli score S_i sono probabilità e quindi saranno tanto maggiori quanto maggiore è la probabilità di trovarci in un dato nodo i alla fine dell'esplorazione casuale del grafo.

Appare quindi naturale ordinare gli score in maniera **crescente** e considerare come **POSITIVI** (appartenenti alla classe considerata) tutti i valori **al di sopra del valore corrente della soglia (cutoff) C** .

Area sotto la curva ROC (AUC)

Per ogni valore assunto dalla soglia C potremo costruire una **matrice di confusione**:

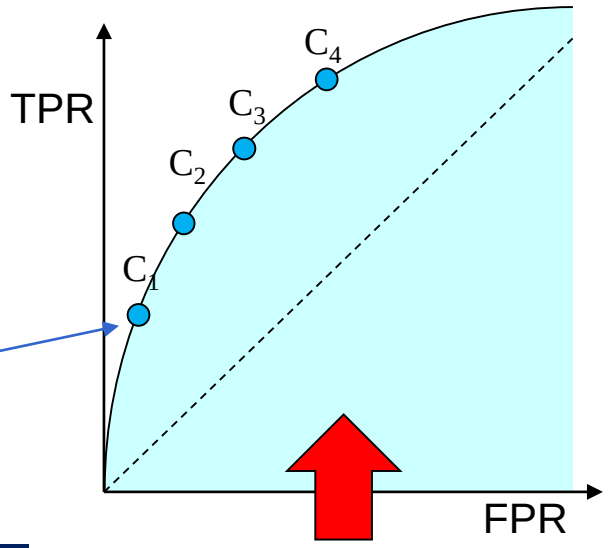
	P	N
pred.P	TP	FP
pred.N	FN	TN

E calcolare:

TPR, o True Positive Rate = TP / (TP+FN)

FPR, o False Positive Rate = FP / (FP+TN)

Questi valori permettono di definire un punto in un piano.



Area sotto la curva ROC (AUC)

AUC vale 1 : ordinamento **perfetto**
AUC vale 0.5 : lancio moneta…

Calcolo AUC

```
# Performance evaluation

library(PerfMeas);

AUC.single(results$p, DB3.labels[,47]);
```