“Università degli studi di Cagliari”
“Facoltà di Ingegneria”

(Anno accademico 2012-2013)

Seminario di Algebra lineare numerica

Text Mining

Teoria e applicazioni numeriche

Candidati:

Alessio Deiana
Riccardo Dessi

Relatore:

Prof. Giuseppe Rodriguez

Indice

Capitolo 1

Capitolo 2

Conclusioni

Capitolo 1

1.1 Introduzione al Text Mining

Nell'ultimo decennio abbiamo assistito a un processo di informatizzazione che ci ha portato nel giro di breve tempo ad essere totalmente dipendenti da una delle più grandi banche dati del mondo: "Internet". Ciò è il frutto di un fenomeno di digitalizzazione che ci ha immerso all'interno di una realtà virtuale tuttora in continua espansione.
Parallelamente a questa evoluzione digitale è sorto però il problema di come gestire questa enorme quantità di dati cercando allo stesso tempo di massimizzare l'efficienza dei metodi di ricerca garantendo agli utenti una navigazione fluida e rapida. A tal proposito una delle branche dell'informatica e della matematica che si occupano di solvere a questo incarico è il *"text mining".*
Volendo essere più rigorosi, si tratta dunque di studiare dei metodi in grado di estrarre delle *informazioni utili,* a partire da una larga e spesso indefinita collezione di testi data una stringa di ricerca. Una delle applicazioni più comuni è la ricerca di documenti scientifici all'interno di una generica libreria virtuale che costituisce l'intero database. Esistono migliaia di esempi in tal senso, basti pensare all'IEEE che offre a tutti i suoi membri la possibilità di poter sfogliare più di cento mila articoli di carattere ingegneristico, o ancora pensiamo al campo medico o informatico o ancora matematico.
E' chiaro che questi metodi non si applicano solo esclusivamente a delle realtà virtuali quali internet; trovano sede infatti su qualsiasi tipo di "sistema informatico" , basti pensare a un comune sistema di gestione per il controllo degli iscritti di una palestra. In questo caso ad esempio la necessità sarebbe quella di poter accedere alla scheda elettronica di ogni singolo cliente per la verifica del pagamento dell'importo mensile, o della validità del certificato medico.
Tutto ciò evidentemente ha anche un impatto economico abbastanza importante, potrà sembrare una banalità, ma con l'avvento dell'archiviazione digitale si sono notevolmente ridotti i costi di gestione dei data-base. Si pensi solo alla quantità di spazio risparmiato, infatti l'equivalente di un archivio cartaceo di 50 m^2 equivale in termini digitali a un server che al massimo arriva ad occupare 2 m^2 di superficie, che si ha i suoi costi di manutenzione, ma niente paragonabile al costo del mattone. Ma il vantaggio più grosso è sicuramente legato all'immediatezza con cui si ha la possibilità di accedere ai documenti utili, consentendo agli operatori un notevole risparmio di tempo. Infine per i più sensibili all'ambiente, si pensi al quantitativo di carta risparmiato, che ha come vantaggio la riduzione del fenomeno del disboscamento e quindi tutti i benefit che ne conseguono. Non a caso i tablet nascono proprio con questa concezione.

1.2 SVD: decomposizione ai valori singolari

La *SVD* (*Singular Value Decomposition*) è una decomposizione di matrici basata sull'uso degli autovalori e degli autovettori. Nonostante la sua apparente semplicità trova applicazione in diversi ambiti dell'algebra numerica e della statistica e risulta molto efficace nella determinazione del rango di una matrice, nella risoluzione di problemi ai minimi quadrati, dove i sistemi da risolvere sono sovradeterminati, nella ricostruzione di immagini sfuocate e affette da rumore e, in generale, nei problemi di *mining*. In quest'ultimo caso stiamo parlando di *SVD troncata* o *TSVD* grazie alla quale è possibile ridurre la dimensione dello spazio dei dati che stiamo trattando.

1.2.1 Definizione della SVD

Questa fattorizzazione può essere applicata a qualsiasi matrice rettangolare A avente dimensione (m,n), scomponendola nel prodotto di tre matrici che presentano proprietà talvolta utili. La *SVD* può essere applicata sia a matrici reali che complesse.

$$A = U \Sigma V^T$$

- La matrice Σ è una matrice pseudo-diagonale avente dimensione (m,m) che contiene i valori singolari della matrice A.
  I valori singolari detti σ_1, σ_2, …, σ_i si calcolano a partire dalle radici degli autovalori della matrice $P=AA^T$ e sono disposti diagonalmente in ordine decrescente. La matrice P è simmetrica e definita positiva quindi i suoi autovalori sono non nulli e le radici degli autovalori sono reali, di conseguenza i valori singolari sono definiti non nulli.

$$\Sigma = \begin{bmatrix} \sigma_1 & \dots & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \sigma_n \\ 0 & \dots & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & 0 \end{bmatrix} se\ m \geq n \qquad \Sigma = \begin{bmatrix} \sigma_1 & \dots & \dots & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \sigma_n & \dots & 0 \end{bmatrix} se\ m \leq n$$

  Il rango delle matrici AA^T, A^TA e A coincidono e si può affermare che il rango delle tre matrici è uguale al numero di autovalori non nulli; questo risultato è interessante in quanto ci restituisce il rango della matrice A.

- Le matrici U e V^T sono matrici ortogonali unitarie e sono dette matrici dei vettori unitari, rispettivamente, destra e sinistra. I vettori colonna della matrice U corrispondono agli autovettori della matrice P.

La matrice Σ è unicamente determinata, cioè i valori singolari di una matrice sono unicamente determinati, mentre le matrici U, V possono essere scelte in diversi modi. Questo si può osservare considerando, per esempio, la matrice identità I che ammette $SVD\,I = V\,I\,V^T$, dove V è una qualunque matrice ortogonale.

1.2.2 Calcolo della SVD con Matlab

E' possibile calcolare la decomposizione ai valori singolari di una matrice ***a*** attraverso Matlab o altri software di calcolo numerico con il comando *svd(a)*. In questo modo l'output risulta essere un vettore colonna contenente i valori singolari di ***a***. Per ottenere la svd completa utilizziamo il comando:

[U,D,V]=svd(a)

```
a=rand(3)

a =

   0.9649   0.9572   0.1419
   0.1576   0.4854   0.4218
   0.9706   0.8003   0.9157

 [U,D,V]=svd(a)

U =

  -0.6208   0.7763  -0.1091
  -0.2820  -0.3509  -0.8929
  -0.7315  -0.5236   0.4368

D =

   2.0818        0        0
        0   0.5707        0
        0        0   0.2592

V =

  -0.6501   0.3252   0.6867
  -0.6324   0.2694  -0.7263
  -0.4212  -0.9064   0.0305
```

Possiamo anche vedere, sempre con l'utilizzo di Matlab, come la matrici U e D siano composte dagli autovettori e dai valori singolari della matrice AA^T.

```
sqrt(eig(a*a'));

ans =

   0.2592
   0.5707
   2.0818

[w,z]=eig(a*a');
w
w =

   0.1091  -0.7763   0.6208
   0.8929   0.3509   0.2820
  -0.4368   0.5236   0.7315
```

1.2.3 Algoritmo SVD

Di seguito viene illustrato uno dei possibili algoritmi per la computazione della svd. I processi utilizzati si basano sulla definizione stessa della *decomposizione ai valori singolari:*

$$A_{m,n}=U_{m,m}\Sigma_{m,n}V^{T}{}_{n,n}$$

$$A^{T}A=V\Sigma^{T}\Sigma V^{T}$$

$$(V^{T})^{-1}=V \quad \rightarrow \quad AV=U\Sigma \quad \rightarrow \quad U=AV\Sigma^{-1}$$

```
function [u,d,v]=mysvd(A)
[m,n]=size(A);
%la matrice d(m,n) è una matrice diagonale costituita dalla radice degli autovalori della
%matrice A'A ; questi sono detti valori singolari e sono disposti in ordine decrescente.
P=A'*A;
av=sort(eig(P),'descend');
d=zeros(n);
for i=1:n
   d(i,i)=sqrt(av(i,1));
end
%la matrice v(n,n) ha per colonne gli autovettori della matrice A'A.
[v,x]=eig(P);
v=[v' eig(P)];
v=sortrows(v,-(n+1));
%i valori singolari della matrice d sono stati disposti in ordine decrescente. Allo stesso modo
%é necessario riordinare gli autovettori delle matrice V rispetto all'ordine decrescente dei
%corrispettivi autovalori; per fare ciò utilizziamo la funzione sortrows.
v=v';
v=v(1:n,1:n);
u=A*v*inv(d);
```

1.3 TSVD: svd troncata

In numerose applicazioni i dati da studiare hanno una dimensione molto elevata e estrapolare da questi delle informazioni utili risulta molto complesso. Per questa ragione si può pensare di approssimare la matrice dei dati originaria con un'altra matrice che abbia minor rango: per fare ciò possiamo utilizzare la TSVD (*Truncated Singular Value Decomposition).*

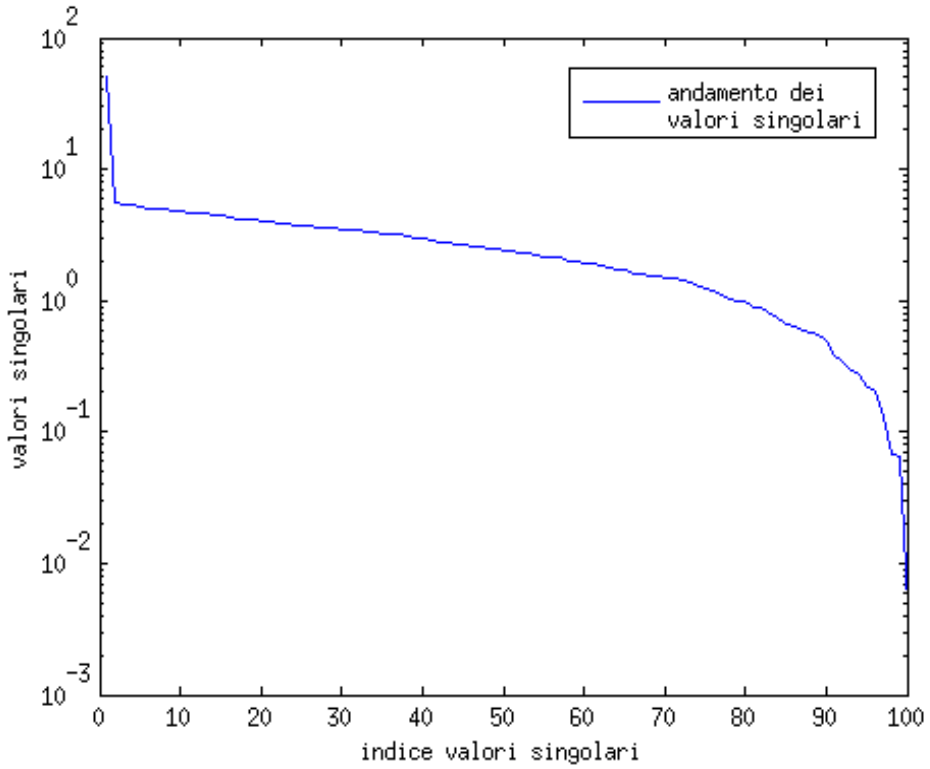
Riscriviamo la svd come:

$$A = U \Sigma V^T = \sum_{i=1}^{n} \sigma_i u_i v_i^T$$

$$A_k = U_k \Sigma_k V_k^T = \sum_{i=1}^{k} \sigma_i u_i v_i^T + \sum_{i=k+1}^{n} \sigma_i u_i v_i^T$$

$$A = A_k + N$$

La *TSVD* cerca di approssimare la matrice A proprio con le sue prime k<n componenti principali A_k, interpretando invece la matrice N come rumore. Questo è possibile a patto di scegliere un adeguato valore k, detto valore di troncamento. Osservando i valori singolari della matrice a notiamo come arrivati a un certo punto avvenga un decadimento del loro valore.
Andando a plottare i valori singolari possiamo notare il caratteristico "andamento a gomito" di questi ultimi. Di seguito viene mostrato un esempio effettuato su una matrice di dimensione 100. L'asse delle ordinate è stato riportato in scala logaritmica:



Nell'ambito del *text mining,* l'algoritmo che lavora con A_k invece che con A, è chiamato *Latent Semantic Indexing (LSI)* perché l'approssimazione di rango basso rivela connessioni che erano nascoste o, appunto, latenti nella matrice A.

Parlando di SVD possiamo definire anche la *Principal Component Analysis (PCA).* Ipotizziamo che X sia una matrice di dati, di dimensione (m,n), rappresentante un insieme di dati. Applicando la *SVD* alla matrice X otteniamo:

$$X = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \ldots + \sigma_i u_i v_i^T$$

I vettori singolari destri v_i sono le componenti direzioni principali di X.
Il vettore $z_1 = X v_1 = \sigma_1 u_1$ ha la varianza più ampia possibile rispetto alla combinazione lineare delle colonne di X. A questo punto dobbiamo trovare il vettore con seconda più ampia varianza, costruito ortogonalmente al primo. Continuando il processo definiamo tutte le componenti principali.
Per mezzo della *TSVD* e della *PCA* effettuiamo quindi una riduzione del numero delle variabili che a questo punto vengono definite "latenti". Ciò avviene tramite una trasformazione lineare delle variabili che proietta quelle originarie in un nuovo sistema nel quale le variabili vengono disposte in ordine decrescente di varianza.

1.4 Introduzione ai comuni algoritmi di Text Mining

L'obbiettivo di questo paragrafo è quello di fornire al lettore una panoramica di alcuni dei principali algoritmi utilizzati nel Text Mining. La caratteristica portante di alcuni di questi metodi è l'accostamento della SVD (motivo per il quale è stata dedicata un'intera sezione a proposito) alla risoluzione di problemi ai minimi quadrati. Nella seconda parte inoltre ci occuperemo di applicare alcuni di questi metodi servendoci di qualche esempio.

1.4.1 Latent Semantic Indexing (LSI)

Uno dei fenomeni di ricerca che accomuna la maggior parte dei data-base è dato dal fatto che alcuni documenti vengono ricercati in modo più frequente rispetto ad altri, che invece presentano un minor numero di ricorrenze.
La LSI sfrutta proprio questo principio, ovvero la possibilità che all'interno di un particolare archivio ci siano delle strutture *"latenti"* che consentono di raggruppare i documenti più richiesti, isolando quelli meno ricercati. Ciò consente di poter proiettare la matrice *termini×documenti* in uno spazio di dimensione minore tramite la SVD troncata.
Se consideriamo A dunque come la matrice *termini×documenti,* allora applicando la SVD troncata otteniamo un'approssimazione di rango *k(fattore di riduzione)* data dalla relazione:

$$A \approx A_k = U_k \Sigma_k U^T = U_k H_k \quad (1.0)$$

La colonna U_k rappresenta il singolo documento a cui è associata una base ortogonale utilizzata per l'approssimazione dell'intero documento. A questo punto se esprimiamo H_k come $H_k=(h_1,h_2,\ldots,h_n)$ *e* riscrivendo inoltre la relazione (1.0) colonna per colonna si ha:

$$a_j \approx U_k h_j$$

Ciò significa che la j-esima colonna di *Hk, hj,* contiene le coordinate del j-esimo documento in termini della base ortogonale U_k. A tal proposito la scelta del valore di riduzione di k non è del tutto così banale e scontata e ovviamente dipende dal modello di riferimento. Tipicamente però questo valore non deve essere elevato, infatti è stato dimostrato sperimentalmente che piccoli valori del fattore di riduzione del rango migliorano notevolmente le prestazioni computazionali dell'algoritmo a discapito però di un errore direttamente proporzionale alle dimensioni della matrice *A.* Questo fenomeno in realtà, se pur confermato dai risultati sperimentali, non è stato ancora dimostrato algebricamente.

1.4.2 Clustering

Nel caso in cui si disponga di un'ampia collezione di documenti, la probabilità che molti di questi trattino argomenti simili è sicuramente elevata. A differenza dunque della LSI in cui si ha la necessità di restringere il campo di scelta con i documenti più ricercati, in questo caso l'obbiettivo è quello di catalogare i documenti per topic, ad esempio isolando tutti i documenti di carattere matematico da quelli medici e così via. Detto in modo brutale, l'intento sarebbe quello di generare tanti insiemi quanti sono il numero degli argomenti all'interno di un intero dominio. Se dunque consideriamo il nostro dominio come uno spazio in $\mathbb{R}^n$dovremo essere in grado di visualizzare al suo interno tutti i sottospazi (cluster) che lo compongono.

Analogamente alla LSI la matrice $C_k \in \mathbb{R}^{m\times k}$(dove anche in questo caso si è fatto uso di un algoritmo per la riduzione del rango) può essere utilizzata come una base che approssima l'insieme in cui si vuole andare a ricercare un determinato documento. Una volta quindi isolato il campo di scelta a partire da un vettore di ricerca, si risolve un problema ai minimi quadrati per estrarre il documento di interesse,

$$min\|A - C_{k^\wedge}\, G_k\| \quad (1.1)$$

in cui si è fatto uso della fattorizzazione QR,

$$C_{k^\wedge} = P_k R, \qquad (P_k\,, R) \in\ \mathbb{R}^{m\times k} \quad (1.2)$$

e quindi sostituendo la relazione (1.2) all'interno della (1.1),

$$min\|A - P_k R\ G_k\| \quad (1.3)$$

A questo punto scrivendo separatamente ciascuna colonna di $A - P_k G_k$che risultano linearmente indipendenti, possiamo applicare i metodi di risoluzione standard:

$$min\left\|a_j - P_{k^\wedge}\, g_j\right\| \quad \text{j} = 1, \dots, \text{n} \quad (1.4)$$

dove g_j corrisponde alle colonne j in G_k. Sino a che P_k contiene colonne ortonormali otterremo valori di $g_j = {P_k}^T a_j$, in questo modo la soluzione della (1.1) diventa:

$$G_{k=}{P_k}^T A \quad (1.5)$$

Lo step successivo consiste nel costruire una query a partire da un vettore di ricerca q la cui dimensione evidentemente dipenderà dal numero di termini in esso presenti; mostreremo in seguito un modo per poterla determinare. Una volta ricavata la query, la soluzione espressa in termini di coseno è data dalla

relazione:

$$cos\theta_j = \frac{q_k T\ g_j}{\|q_k\|\ \|g_j\|} \qquad -1 \leq cos\theta_j \leq 1 \quad (1.6)$$

1.4.3 La fattorizzazione non negativa (NMF)

Un altro modo di vedere le cose è quello di approssimare la matrice A utilizzando la NMF:

$$A = WH,\ \ W \geq 0, \quad H \geq 0,$$

dove $(W, H) \in \mathbb{R}^{m \times k}$. Le colonne j-esime contengono in prima approssimazione le cordinate del documento j, la base (in questo caso non ortogonale) è rappresentata dalle colonne di W. Come fatto precedentemente vorremo determinate una rappresentazione per il vettore di ricerca q utilizzando la stessa base e risolvendo un problema ai minimi quadrati del tipo $min\|q - W\ \hat{q}\|$. Procedendo dunque con la fattorizzazione QR della matrice W risulta:

$$W = QR, \qquad (P, R) \in \mathbb{R}^{m \times k},$$

e la query corrispondente alla base approssimata è data dalla relazione:

$$\hat{q} = R^{-1}Q^T q \ (1.7)$$

da cui è immediato ricavare la soluzione espressa, come nel caso precedente, in termini di coseno:

$$cos\theta_j = \frac{q_k T\ h_j}{\|q_k\|\ \|h_j\|} \quad -1 \leq cos\theta_j \leq 1 \qquad (1.8)$$

1.5 La matrice dei termini x documenti

Nei paragrafi precedenti abbiamo dato per scontato che la matrice A e il vettore di ricerca fossero già noti. In realtà il processo di conversione in termini matriciali non è affatto un'operazione banale. Ora questo argomento esula dalle finalità di questo testo, ma allo stesso tempo il nostro intento è quello di lasciare al lettore una visione a 360° del problema, motivo per il quale spenderemo qualche parola senza entrare particolarmente nel dettaglio.

Il riconoscimento delle parole nei testi tipicamente è affidato ad algoritmi scritti nei più comuni linguaggi di programmazione; prendono in ingresso un certo

frammento di testo e in uscita restituiscono un vettore sparso, cioè caratterizzato dalla presenza di uni e zeri. Il testo viene quindi confrontato parola per parola con un dizionario interno, in cui sono presenti tutte le parole che devono essere riconosciute. Se dunque viene riscontrato nel documento una parola che è presente anche nel dizionario, l'elemento del vettore corrispondente assumerà un valore *logico* uno, viceversa zero. Una struttura che consente tale operazione è ad esempio : la *stringcompare* del C, essa prende in ingresso 2 stringhe e le confronta restituendo quindi in uscita o uno o zero. E' immediato dunque constatare che la dimensione del vettore colonna in uscita, può assumere dimensioni veramente importanti che dipenderà dal numero delle parole presenti nel dizionario. Vediamo a questo punto un esempio.

Siano dati i seguenti documenti:

d1 : Romeo and Juliet.

d2 : Juliet: O happy dagger!

d3 : Romeo died by dagger.

d4 : "Live free or die", that's the New-Hampshire's motto.

d5 : Did you know, New-Hampshire is in New-England.

Sia inoltre assegnato un dizionario, composto dalle seguenti parole:

Romeo ,Juliet, happy, dagger, live, die, free, new-hampshire.

Se ad esempio decidiamo di confrontare il documento *d1,* allora il vettore di uscita corrispondente sarà:

	d1
$romeo$	1
$juliet$	1
$happy$	0
$dagger$	0
$live$	0
die	0
$free$	0
$new-hampshire$	0

ripetendo infine il procedimento per tutti gli altri documenti otteniamo la matrice A. Come si può osservare la dimensione delle righe è data dal numero dei termini presenti all'interno del dizionario, mentre la dimensione delle colonne corrisponde al numero dei documenti.

	d_1	d_2	d_3	d_4	d_5
romeo	1	0	1	0	0
juliet	1	1	0	0	0
happy	0	1	0	0	0
dagger	0	1	1	0	0
live	0	0	0	1	0
die	0	0	1	1	0
free	0	0	0	1	0
new-hampshire	0	0	0	1	1

Chiaramente tanto maggiore sarà la dimensione di A, tanto maggiore sarà l'incidenza del carico computazionale offerta dall'algoritmo. Possono però essere prese delle precauzioni che ci consentono di ridurre il numero di righe, che ricordiamo è legato al numero di parole presenti nel dizionario di riconoscimento. Per esempio si può pensare di costruire un dizionario fatto solo ed esclusivamente da *"parole radice"* per cui se viene riscontrata una parola con la stessa radice quest'ultima verrà assegnato il valore logico uno. Ciò consentirebbe di ridurre enormemente le dimensioni della matrice (in termini di righe).

Parola radice	***Parole riscontrabili***
vol	**vol**are
	volume
	volontà
	volto
	voltare
	volgo
	vortice
tor	**tor**o
	torre
	torna
	tornare
	torta
	torcia
	torrida

Capitolo 2

2.1 Introduzione alla parte sperimentale

Prima di addentrarci nella parte sperimentale vera e propria è bene definire un banco di prova che utilizzeremo nei due algoritmi successivi.
Siano dati quindi i seguenti documenti:

*Documento 1: The **Google matrix** P is a model of the **Internet.***
*Documento 2: P_{ij} is non zero if there is a **link** for **webpage** j to i.*
*Documento 3: The **Google matrix** is used to **rank** all **web pages**.*
*Documento 4: The **ranking** is done by solving a **matrix eigenvalue** problem.*
*Documento 5: **England** dropped out of the top tan in the **FIFA ranking**.*

Le parole evidenziate in neretto corrispondono alle parole presenti nel dizionario di riconoscimento. Utilizzando lo stesso procedimento descritto nel pargrafo 1.5 ,costruiamo la matrice sparsa A, che molto semplicemente si presenta nella forma:

Termini	Doc1	Doc2	Doc3	Doc4	Doc4
Eigenvalue	0	0	0	1	0
England	0	0	0	0	1
FIFA	0	0	0	0	1
Google	1	0	1	0	0
Internet	1	0	0	0	0
Link	0	1	0	0	0
Matrix	1	0	1	1	0
Page	0	1	1	0	0
Rank	0	0	1	1	1
Web	0	1	1	0	0

L'altro elemento di interesse è definire un vettore di ricerca. Ipotizziamo di voler ricercare all'interno del nostro campo di scelta, i documenti più chiusi rispetto a una query così fatta: "***ranking of web pages***".
Esso corrisponderà a un vettore colonna dove i termini non nulli corrispondono agli elementi che si vogliono rilevare in A:

$$q' = (0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1) \in \mathbb{R}^{10}$$

Ovviamente quest'ultimo, a differenza della matrice A, non deve essere fissato a priori, pertanto la scelta risulta assolutamente arbitraria. Sembra dunque evidente che Il nostro obbiettivo è quello di trovare i documenti che più si

avvicinano alla query scelta e quindi classificarli in base alla loro rilevanza. In questo caso si è scelto un modello per cui è immediato riscontare che il Doc 3 è quello che più si avvicina al vettore di ricerca. Così facendo saremo in grado di poter constatare facilmente la funzionalità degli algoritmi esaminati. E' importante però specificare che nella realtà dei fatti si va a operare con matrici di dimensioni ben maggiori; si pensi alla matrice di Google che con molta facilità raggiunge ordini di 10^6 (righe, colonne).

2.2 Esperimento Latent Semantic Indexing (LSI)

Il primo esperimento consiste nell'applicazione della LSI (vedi 1.4.1). Come spiegato precedentemente questo algoritmo rileva la presenza di strutture latenti in A, quest' ultime saranno tanto più evidenti quanto più è adeguato il valore del fattore di troncamento . In realtà, data la dimensione limitata di A, questo fenomeno non sarà così visibile, infatti se andiamo a plottare i valori singolari ottenuti fattorizzando con la SVD non otteniamo la tipica *caratteristica a gomito*, quindi utilizzeremo un altro criterio di valutazione*.
Il primo passo dell'algoritmo consiste nella fattorizzazione SVD della matrice A:

V =

2.8546	0	0	0	0
0	1.8823	0	0	0
0	0	1.7321	0	0
0	0	0	1.2603	0
0	0	0	0	0.8483
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

*E' immediato constatare che l'informazione risiede maggiormente nei primi due/tre fattori singolari. Se assumiamo dunque un k pari a 2, ne consegue la seguente fattorizzazione troncata.

S =

-0.1425	0.2430
-0.0787	0.2607
-0.0787	0.2607
-0.3924	-0.0274
-0.1297	0.0740
-0.1020	-0.3735
-0.5348	0.2156
-0.3647	-0.4749
-0.4838	0.4023
-0.3647	-0.4749

V =

2.8546	0
0	1.8823

D =

-0.3702	0.1393
-0.2912	-0.7030
-0.7498	-0.1909
-0.4068	0.4573
-0.2246	0.4908

Ricaviamo ora la matrice dei *termini* definita come: $T = S_2\Sigma_2$,e la matrice dei documenti a sua volta definita come: $H = \Sigma_2 {U_2}^T$, in altri termini $T = SV$ *e* $H = VD^T$.

T=	H =
-0.4068 -0.4573 -0.2246 -0.4908 -0.2246 -0.4908 -1.1200 0.0516 -0.3702 -0.1393 -0.2912 0.7030 -1.5268 -0.4058 -1.0410 0.8938 -1.3812 -0.7573 -1.0410 0.8938	-1.0569 -0.8314 -2.1404 -1.1612 -0.6412 -0.2622 1.3232 0.3592 -0.8608 -0.9238

Volendo riscrivere e isolare i termini corrispondenti alle righe della matrice T otteniamo:

$$eigenvalue = \begin{bmatrix} 0.4068 \\ -0.4573 \end{bmatrix} \qquad link = \begin{bmatrix} -0.2912 \\ 0.7030 \end{bmatrix}$$

$$England = \begin{bmatrix} -0.2246 \\ -0.4908 \end{bmatrix} \qquad matric = \begin{bmatrix} -1.5268 \\ -0.4058 \end{bmatrix}$$

$$FIFA = \begin{bmatrix} -0.2246 \\ -0.4908 \end{bmatrix} \qquad page = \begin{bmatrix} -1.0410 \\ 0.8938 \end{bmatrix}$$

$$Google = \begin{bmatrix} -1.1200 \\ 0.0516 \end{bmatrix} \qquad rank = \begin{bmatrix} -1.3812 \\ -0.7573 \end{bmatrix}$$

$$Internet = \begin{bmatrix} -0.3702 \\ -0.1393 \end{bmatrix} \qquad web = \begin{bmatrix} -1.0410 \\ 0.8938 \end{bmatrix}$$

Applicando lo stesso procedimento per la matrice H, isolando le colonne corrispondenti ai singoli documenti si ottiene:

$$doc1 = \begin{bmatrix} -1.0569 \\ -0.2622 \end{bmatrix}$$

$$doc2 = \begin{bmatrix} -0.8314 \\ -1.3232 \end{bmatrix}$$

$$doc3 = \begin{bmatrix} -2.1404 \\ 0.3592 \end{bmatrix}$$

$$doc4 = \begin{bmatrix} -1.1612 \\ -0.8608 \end{bmatrix}$$

$$doc5 = \begin{bmatrix} -0.6412 \\ -0.9238 \end{bmatrix}$$

Questa nuova rappresentazione ci consentirà di poter applicare le relazioni successive in modo più ordinato e leggibile.
Lo step successivo consiste nell'esprimere q come la media dei termini che lo compongono, il che significa sommare i vettori colonna corrispondenti a: *rank, web, page* e dividerli per 3:

$$q = \frac{\begin{bmatrix}-1.0410\\ 0.8938\end{bmatrix} + \begin{bmatrix}-1.3812\\ -0.7573\end{bmatrix} + \begin{bmatrix}-1.0410\\ 0.8938\end{bmatrix}}{3} = \begin{bmatrix}-1.1544\\ 0.3435\end{bmatrix}$$

A questo punto abbiamo tutti gli elementi per poter determinare il valore della soluzione che in termini di coseno si presenta nella forma:

$$cos\theta_j = \frac{qT\ h_j}{\|q_k\|\ \|h_j\|} \quad (2.0)$$

Quello che uscirà fuori sarà dunque un vettore la cui dimensione dipenderà dal numero dei documenti, nel nostro caso cinque. Il che significa ripetere l'operazione tante volte quante sono le colonne della matrice H. Pertanto al passo 1 sostituendo i rispettivi valori all'interno della (2.0) otteniamo:

$$cos\theta_1 = \frac{[-1.1544 \quad 0.3435]\begin{bmatrix}-1.0569\\ -0.2622\end{bmatrix}}{1.2044 \times 1.0889} = 0.8616$$

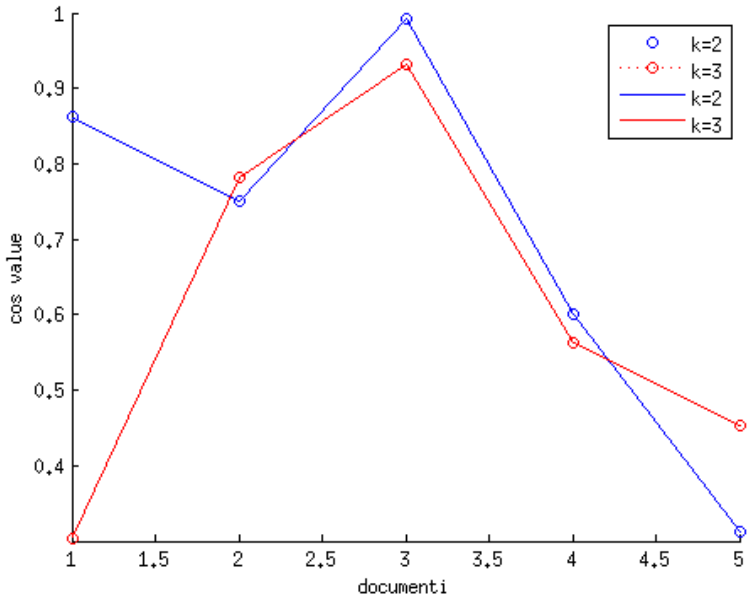
Se ripetiamo tale operazione sino a espletare tutte le colonne di H il vettore cos risulterà uguale:

Termini	Doc1	Doc2	Doc3	Doc4	Doc5
$cos\theta_j$	0.8616	0.7514	0.9925	0.6001	0.3123

E' immediato constatare come la soluzione offerta dal doc3 è quella che più si avvicina al valore 1, osserviamo però che anche la soluzione associata al doc1 presenta un valore prossimo all'unità. Se andiamo a verificare nella matrice A, il doc1 contiene solo uno degli elementi presenti in q. Questo ci suggerisce che il valore di k scelto precedentemente risulta troppo basso per avere una buona linearità nella soluzione. Infatti ponendo questa volta k=3 si ottengono dei valori più conformi al modello reale:

Termini	Doc1	Doc2	Doc3	Doc4	Doc5
$cos\theta_j$	0.3027	0.7812	0.9311	0.5630	0.4524

Il tutto risulta ancora più evidente se plottiamo entrambe le soluzioni in funzione del numero dei documenti. Osservando la linea rossa (corrispondente al valore di k=3) si può constatare come la soluzione converga in modo molto più lineare.



Riassumiamo dunque i passi principali dell'algoritmo:

1. *Fattorizzazione ai valori singolari della matrice termini x documenti.*
2. *Scelta del valore di troncamento k.*
3. *calcolo di q.*
4. *Costruzione del vettore* $cos\theta_j = \frac{qT\ h_j}{\|q_k\|\ \|h_j\|} \in \mathbb{R}^{documenti}$

Il nostro intento a questo punto è costruire una funzione *matlab* che presa in ingresso la matrice A, il vettore q e il fattore di troncamento restituisca in output il vettore *cos.* Sicuramente la parte che necessita un minimo di ragionamento è quella legata alla costruzione della query da applicare alla relazione (2.0) (che ci ha fatto dannare!). Si può pensare di estrapolare il numero e le coordinate delle componenti non nulle del vettore q, e quindi costruire una matrice *v* che contiene i termini t_i di interesse (*rank, web, page*). Una possibile implementazione è riportata di seguito:

```
function[b,cos,H,T,out]=LSI(A,q,k)
tic
[row,col]=size(A);
[S,V,D]=svds(A,k);
T=S*V;
H=V*D';
```

```
[c,r]=size(H);
[l,j,s]=find(q);
[x,m]=size(s);

for u=1:k
   for i=1:m
      v(i,u)=T(j(i),u);
   end
end
b=sum(v)/m;
for a=1:r
   cos(a)=(b*H(1:k,a)/(norm(H(1:k,a))*norm(b)));
end
[c,id]=max(cos);
out=A(1:row,id);
toc
end
```

Per completezza riportiamo l'output dello script. Gli ingressi A,q, k coincidono con quelli utilizzati nell'esempio argomentato precedentemente.

```
[query,cos,H,T,output]=LSI(A,q',2)
Elapsed time is 0.009488 seconds.

query =

  -1.1544   -0.3435


cos =

   0.8616    0.7514    0.9925    0.6001    0.3123


H =

  -1.0569   -0.8314   -2.1404   -1.1612   -0.6412
   0.2622   -1.3232   -0.3592    0.8608    0.9238


T =

  -0.4068    0.4573
  -0.2246    0.4908
  -0.2246    0.4908
  -1.1200   -0.0516
  -0.3702    0.1393
```

```
  -0.2912   -0.7030
  -1.5268    0.4058
  -1.0410   -0.8938
  -1.3812    0.7573
  -1.0410   -0.8938


output =

    0
    0
    0
    1
    0
    0
    1
    1
    1
    1
```

In uscita dunque viene riportato il valore di *q (query)* che come potrete constatare è identico a quello che ci siamo calcolati manualmente eccetto per il fatto che si presenta nella forma già trasposta. Di seguito vengono riportate le matrici T, H e il vettore *cos* e anche in questo caso fortunatamente i valori coincidono. Infine *out* restituisce il vettore colonna della matrice A che meglio è approssimata dalla query prescelta, che corrisponde proprio al Doc3 (se non vi fidate andate a ricontrollarvi la matrice A). Ci teniamo a precisare che questa è una funzione scritta e ideata dai compositori di questo testo, e che quindi sicuramente presenta dei limiti computazionali; esistono sicuramente delle versioni ottimizzate che di fatto riescono a fare la stessa cosa ma in tempi e con precisione molto maggiore. Pertanto rimandiamo tutti coloro che desiderano approfondire l'argomento ad altri testi più specifici.

2.3 Esperimento Zero Permutation (ZP)

Premettiamo che l'algoritmo trattato di seguito è totalmente ideato dagli autori di questo testo, vuol essere un modo alternativo per sperimentare sui metodi del Text Mining senza fare uso fattorizzazioni o relazioni algebriche particolari (d'altronde siamo studenti di ingegneria e ci piacciono le cose semplici e pratiche).
Anche in questo caso faremo uso del banco di prova dell'esempio precedente in modo tale da poter confrontare i risultati ottenuti. L'elemento cardine dell'algoritmo ZP, come da titolo, consiste nel valutare il numero di permutazioni degli elementi non nulli della colonna corrispondente al documento a_j rispetto al vettore di ricerca. Ovviamente questo metodo non può

tenere conto della presenza di strutture latenti all'interno della matrice A che rimane invariata nel corso di tutta l'esecuzione dell'algoritmo.
Di seguito è riportato uno pseudo codice dell'algoritmo:

```
count(n)=0;
for i=1,...,n
  for j=1,...,m
    if a_ji=1
      r_ji= a_ji-q_j;
        if  r_ji=0
          count_i = count_i + 1;
        else count_i= count_i ;
    else  j+1
```

Prima di tutto inizializziamo a zero un vettore *count* che ha la dimensione corrispondente alle colonne di *A*. Successivamente valutiamo il valore dell'elemento a_{ji}: se quest'ultimo vale 1 allora verrà confrontato con l'elemento corrispondente q_j , il che significa semplicemente effettuare un'operazione di sottrazione membro a membro; se il risultato di tale operazione è uguale a zero, allora significa che abbiamo riscontrato una permutazione e quindi l'elemento *count* i-esimo viene incrementato, altrimenti resta invariato; in caso contrario ($a_{ji}=0$), si salta al valore j-esimo successivo. Questo perché l'elemento a_{ji}, in quanto nullo, anche se l'elemento q_j fosse pari a 1,non avrebbe senso computare l'operazione di confronto in quanto siamo sicuri che il termine j-esimo non è contenuto nella colonna del documento in questione. Così facendo costruiremo un vettore *count* che per ogni elemento conterrà il numero delle permutazioni di ogni singola colonna, e questo viene fatto costruendo la *matrice di confronto R*.
Per chiarimento riportiamo un passo dell'algoritmo.

Siano noti la matrice A e il vettore q :

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \qquad q = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

quindi procediamo con il confronto della terza colonna con il vettore di ricerca ottenendo così la terza colonna della matrice R, e allo stesso tempo contiamo il numero delle permutazioni

$$r_{j,1} = a_{j,3} - q = \begin{bmatrix}0\\0\\0\\1\\0\\0\\1\\1\\1\\1\end{bmatrix} - \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\1\\1\\1\end{bmatrix} = \begin{bmatrix}0\\0\\0\\1\\0\\0\\1\\0\\0\\0\end{bmatrix} \quad \begin{matrix}\\\\\\\text{count+1}\\\text{count+1}\\\text{count+1}\\\\\\\end{matrix} \qquad count[3] = 3$$

Ripetendo il procedimento per tutte le colonne di A, la rappresentazione della matrice *R* completa e del vettore *count* ad essa associata sarà:

$$R = \begin{bmatrix}0&0&0&1&0\\0&0&0&0&1\\0&0&0&0&1\\1&0&1&0&0\\1&0&0&0&0\\0&1&0&0&0\\1&0&1&1&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix} \qquad count = \begin{bmatrix}0 & 2 & 3 & 1 & 1\end{bmatrix}$$

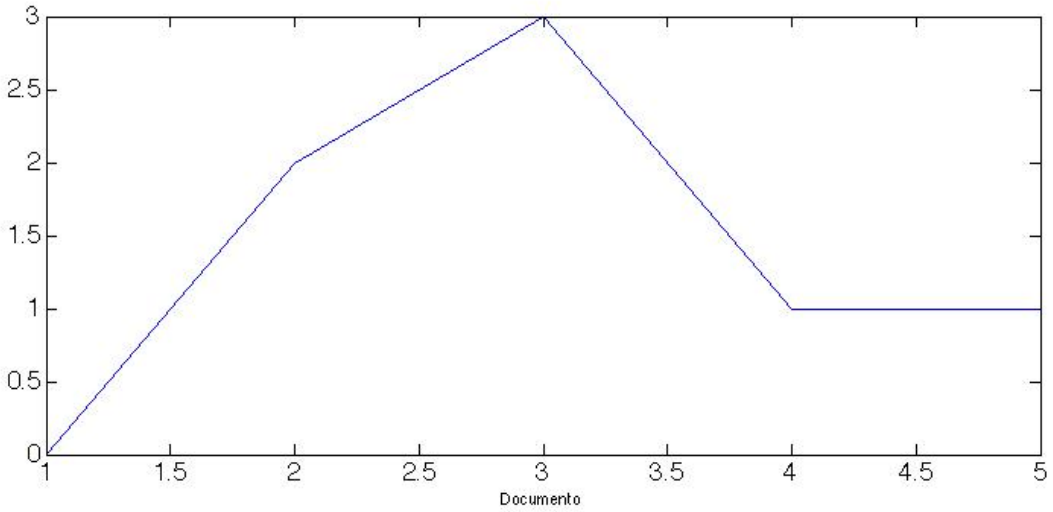
Anche in questo caso il Doc3 è il documento che più si avvicina alla query selezionata, infatti l'elemento *count* corrispondente è quello che contiene il maggior numero di permutazioni. Per completezza riportiamo una possibile implementazione del codice in *matlab* che abbiamo utilizzato per le diverse prove:

```
function [id,c,r,count,out]= search (A,q)

tic
 [m,n]=size(A);
count(n)=0;
for i=1:n
   for j=1:m
      if A(j,i) == 1;
         r(j,i)=A(j,i) - q(j);
```

```
      if  r(j,i)==0 count(i)=count(i) +1;
      else
      count(i)= count(i);
      end
      else j+1;
      end
   end
end
[c,id]=max(count);
toc
out=A(1:m,id);
end
```

In fine riportiamo anche l'output:

```
[id,c,r,count,out]=search(A,q)
Elapsed time is 0.000109 seconds.

id =

     3
 c =

     3

r =

     0     0     0     1     0
     0     0     0     0     1
     0     0     0     0     1
     1     0     1     0     0
     1     0     0     0     0
     0     1     0     0     0
     1     0     1     1     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0

 count =

     0     2     3     1     1
 out =

     0
     0
     0
     1
     0
     0
     1
     1
     1
     1
```



Come si può osservare l'andamento della soluzione pare essere simile a quella valutata precedentemente con la LSI per il valore di k=3.

Conclusioni

I due algoritmi implementati sfruttano principi e logiche totalmente differenti ma entrambi portano allo stesso risultato. Tuttavia il primo algoritmo ha il vantaggio di stimare la matrice A e identificarne possibili strutture latenti, pertanto risulta essere molto più completo rispetto al metodo ZP che sfrutta logiche meccaniche (non fa altro che confrontare termine per termine). Tuttavia ZP presenta dei vantaggi computazionali importanti, infatti esegue semplici operazioni algebriche, e tutto il carico è concentrato sulle condizioni (if), mentre la LSI, proprio per la presenza della fattorizzazione SVD, risulta computazionalmente più pesante. A conferma di quanto appena detto, se confrontate i tempi (che trovate nei due output precedentemente mostrati) potete osservare come il tempo di esecuzione ZP sia circa due ordini di grandezza inferiore rispetto a quello della LSI, e il fenomeno si ripete anche per valori di dimensione della matrice A elevati. Risulta comunque un azzardo poter pensare di confrontare i due metodi, anche se di fatto svolgono la stessa funzione, in quanto partono da presupposti totalmente differenti. A tal proposito sembra quasi che ci sia da parte nostra l'intenzione di voler screditare ZP, in realtà siamo convinti che anche un algoritmo così semplice potrebbe trovare un piccolo spazio all'interno dello sconfinato mondo del Text Mining, ad esempio andrebbe benissimo per un piccolo sistema informatico che si deve occupare di gestire un numero limitato di documenti, in cui magari non si ha la necessità di dover individuare strutture latenti o catalogare per argomento(clustering). In conclusione c'è ancora tanto da scoprire e inventare motivo per il quale il Text Mining continuerà ad essere ancora per molto tempo uno degli oggetti di studio più importanti della matematica e informatica moderna.