Laboratorio Sperimentale di Matematica Computazionale 2016-2017

Parte I

Paolo Giordano

26 luglio 2017

INDICE

1 LEZIONE II, 3 MARZO 2017

ESERCIZIO 1

La funzione *graph_laplacian* è:

```
function L=graph_laplacian(A)
%A e' la matrice di adiacenza di un grafo non orientato
%L e' la matrice Laplaciana del grafo descritto da A
d=size(A);
a=ones(d(1),1);
L=spdiags(A*a,0,d(1),d(1))-A;
```

Punto a

Moltiplicando la matrice L per il vettore $(1,1,\ldots,1)^T$ si ottiene il vettore nullo, che ha di conseguenza norma nulla; infatti, dando il comando

```
>> norm(L*ones(60,1))
```

si ottiene

```
ans=

        0
```

Sia

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots \\ a_{21} & a_{22} & \ldots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

Per dimostrare che $(1,1,\ldots,1)^T$ è autovettore del Laplaciano qualunque sia il grafo $\mathcal{G}$, è sufficiente scrivere L come

$$L = D - A = \begin{pmatrix} \sum\limits_{\substack{i=1 \\ j\neq 1}} a_{ij} & -a_{12} & -a_{13} & \ldots \\ -a_{21} & \sum\limits_{\substack{i=2 \\ j\neq 2}} a_{ij} & -a_{23} & \ldots \\ -a_{31} & -a_{32} & \sum\limits_{\substack{i=3 \\ j\neq 3}} a_{ij} & \ldots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

e moltiplicare questa matrice per $(1, 1, \ldots, 1)^T$, in questa maniera si ottiene un vettore che ha componente $k - esima$ uguale a

$$\sum_{\substack{i=k \\ j \neq k}} a_{ij} + \sum_{\substack{i=k \\ j \neq k}} -a_{ij},$$

che è uguale a zero per ogni k.

Punto b

La matrice L è simmetrica poichè si ottiene sottraendo ad una matrice diagonale una matrice simmetrica ($L = D - A$). Per verificare che i suoi autovalori siano tutti maggiori o uguali di zero, si può usare la funzione *eigs* nel modo seguente:

```
>> eigs(L,2,'SM')
```

il quale ci dice che i due autovalori più piccoli sono:

```
ans =

   1.61624894705837e-01
  -2.22044604925031e-17
```

ans è formato da due elementi: il primo maggiore di zero e l'altro minore della precisione di macchina e per questo approssimabile a zero. Essendo questi i due autovalori più piccoli, siamo sicuri che tutti gli altri siano maggiori di zero.

È possibile dimostrare che questa è una proprietà generale della matrice Laplaciana, infatti, applicando il primo teorema di Gerschgorin alla matrice L, si osserva che i cerchi di Gerschgorin hanno tutti centro positivo e raggio uguale alla distanza del centro dall'origine degli assi. (I centri dei cerchi sono tutti positivi perchè gli elementi sulla diagonale di L sono somme di elementi della matrice A, che è composta da soli 0 e 1.)

Punto c

Diamo il comando

```
>> eigs(L,2,'SM')
```

il quale da come risposta

```
ans=

        0.243401746139933
        0.000000000000000
```

da cui si ottiene che il secondo autovalore più piccolo di L è

$$\lambda_2 = 0.243401746139933$$

ESERCIZIO 2

La funzione *Fiedler_vector* è:

```
function [l,x,perm]=Fiedler_vector(A)
% A matrice di Adiacenza di un grafo
% x e' il vettore di Fiedler
% l e' il secondo autovalore piu' piccolo di L
% perm e' la permutazione che ordina x in ordine crescente

L=graph_laplacian(A); %Calcolo il Laplaciano di A
[V,D]=eigs(L,2,'SM');    %Calcolo autovalore
                         %ed autovettore di Fiedler
l=D(1,1);
x=V(:,1);
[y,perm]=sort(x);     %Calcolo la permutazione che ordina x
B=A(perm, perm);      %Riordino A in base alla permutazione perm
spy(B);                  %Visualizziamo la struttura
                         %della matrice B
[C,d]=spdiags(B);     %Calcolo l'ampiezza di banda di B e di A
band_temp=size(d);
band1=band_temp(1);
[C,d]=spdiags(A);
band_temp=size(d);
band2=band_temp(1);
if band1<band2    %Verifico se la banda di B
                   %e' minore di quella di A
    disp('La banda di B e' minore di quella di A.')
end
```

Per il calcolo del Laplaciano di A si è usata la funzione *graph_laplacian* dell'esercizio 1.
Tramite la funzione *spdiags* otteniamo un vettore *d* che ha dimensione pari al numero di diagonali non nulle di una matrice sparsa, ovvero il suo numero di banda.

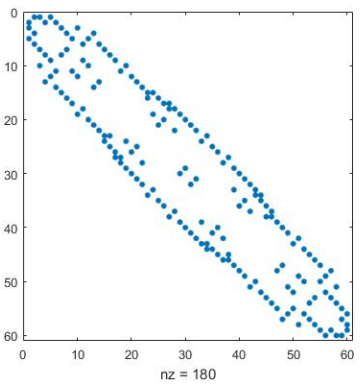


Figura 1: Grafico della struttura della matrice *B*, ottenuto con la funzione *spy*.

ESERCIZIO 3

Punto a

La funzione *incidence_matrix* è:

```
function M=incidence_matrix(A)
% A matrice di adiacenza di un grafo
% M sua matrice di incidenza

%Le diagonali le rendo uguali a zero
%Controlliamo se A e' simmetrica
if A==A'
    %Se A e' simmetrica consideriamo la sua triangolare superiore
    A1=triu(A)-diag(diag(A));
    %numero di elementi di A1 non nulli
    d=size(A);
    count=nnz(A1);
    M=zeros(d(1),count);
    column=1;
    %Creo la matrice M
    for k=1:d(1)
        for j=k:d(1)
            if A1(k,j)~=0
                M(k,column)=1;
                M(j,column)=-1;
                column=column+1;
            end
        end
    end
else
    %Se A non e' simmetrica la considero tutta
    A1=A-diag(diag(A));
    d=size(A);
    count=nnz(A1);
    M=zeros(d(1),count);
    column=1;
    %Creo la matrice M
    for k=1:d(1)
        for j=1:d(1)
            if A1(k,j)~=0
                M(k,column)=1;
                M(j,column)=-1;
                column=column+1;
            end
        end
    end
end
```

Punto b

Assegniamo la matrice *bucky* ad A:

```
>>[A,v]=bucky;
```

Calcoliamo la Laplaciana di A:

```
>>L=graph_laplacian(A);
```

Verifichiamo che si ha $L = M * M^T$:

```
>>M=incidence_matrix(A);
>>if L==M*M'
        disp('L=M*M^T')
    end
```

L'output è:

```
L=M*M^T
```

Osserviamo inoltre che

```
>> size(M)

ans=

       60     60
>> rank(M)

ans=

        59
```

cioè che *rank(M)=n-1*.

ESERCIZIO 4

La funzione *solve_augmented* è:

```
function [x]=solve_augmented(M,b)
    % USO: x=solve_augmented(M, b),
    % M matrice di incidenza di un grafo,
    % b vettore dei termini noti del sistema lineare Lx=b
    [n,m]=size(M);
    B=sparse([eye(m) M';M zeros(n)]); %B deve essere sparsa
    [Q,R,P]=qr(B); %utilizziamo il metodo qr con pivot sulle
    % colonne per calcolare una fattorizzazione
    % "rank-revealing" cioe' dove le ultime t righe di R sono
    %nulle se rank(R)=n-t;
    c1=rand(m,1);
    c2=M*c1-b;
    c=[c1;c2];
    z=Q\c; %soluzione del sistema Qz=c
    soglia=0.0000000001;
    r=sum(abs(diag(R))>soglia); % rango di R
    if abs(z(r+1:n+m))>10^-12*norm(z) || r==n+m
    %controlla che z(r+1:n+m)==0 e che R sia singolare
    error('qualcosa va male, vettore b non adeguato?');
    end
    R1=R(1:r,1:r); %minore principale di testa di R di rango pieno
    z1=z(1:r);
    V1=R(1:r,r+1:end); %prime r righe delle ultime colonne
                       %della matrice R
    alpha=ones(1,n+m-r); % vettore qualsiasi di
                         %dimensioni opportune...
    w1=R1\(z1-V1*alpha);
    w=[w1;alpha];
    y=P*w; % occorre permutare le componenti di w
    x=y(m+1:end);
    % verifica che x sia soluzione di Lx=b
end
```

TEST DELLE FUNZIONI

Test delle funzioni tramite *testL2*:

```
>>testL2
Laplacian test passed
Incidence matrix test passed
La banda di B e' minore di quella di A.
Fiedler test passed
Augmented solution test passed
```

Dove l'output *"La banda di B è minore di quella di A."* è dovuto all'esecuzione della funzione *incidence_matrix*, la quale inoltre restituisce, tramite funzione *spy*, il seguente grafico:

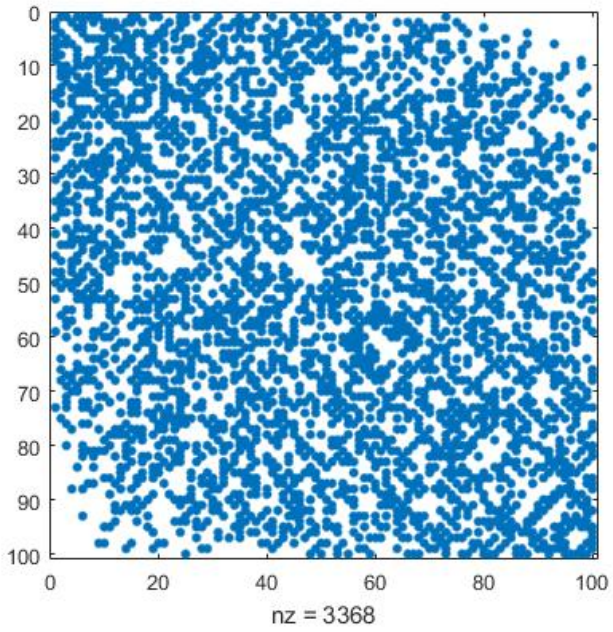


Figura 2: Output di *spy* della funzione *incidence_matrix* durante il *testL2*.

2 LEZIONE III, 8 MARZO 2017

ESERCIZIO 1

La funzione *mean_classifier* è:

```
function [class]=mean_classifier(TrainingSet,DigitsTraining...
   ,TestSet)
% TrainingSet: una matrice mxn contente m immagini su n vettori
% DigitsTraining: vettore di n componenti che riporta la cifra
% rappresentata di ogni immagine in TraininSet
% TestSet: matrice di immagini da classificare
% class: vettore che conterra' la
%classificazione delle cifre nel TestSet
M=zeros(256,10);
cont=zeros(1,10);
%leggiamo le occorrenze nel vettore DigitsTraining
%e creiamo la matrice M
for i=1:1707
    cont(DigitsTraining(i)+1)=cont(DigitsTraining(i)+1)+1;
    M(:,DigitsTraining(i)+1)=M(:,DigitsTraining(i)+1)+...
         TrainingSet(:,i);
end
for i=1:10
    M(:,i)=M(:,i)/cont(i);
end
%mostriamo a schermo le immagini delle colonne di M
for c=1:10
     subplot(2, 5, c)
     ima2(M(:, c))
end
%classifichiamo le cifre di TestSet tramite le medie
%contenute nella matrice M
for i=1:2007
    v=zeros(1,10);
    for k=1:10
        v(k)=norm(TestSet(:,i)-M(:,k));
    end
    for j=1:10
        if v(j)==min(v)
            class(i)=j-1;
        end
    end
end
```

Salviamo nel vettore *class* la classificazione delle cifre nel TestSet:

```
>>load dataset_digits
```

```
>>[class]=mean_classifier(TrainingSet, DigitsTraining, TestSet)
```

L'output della funzione *ima2* è:

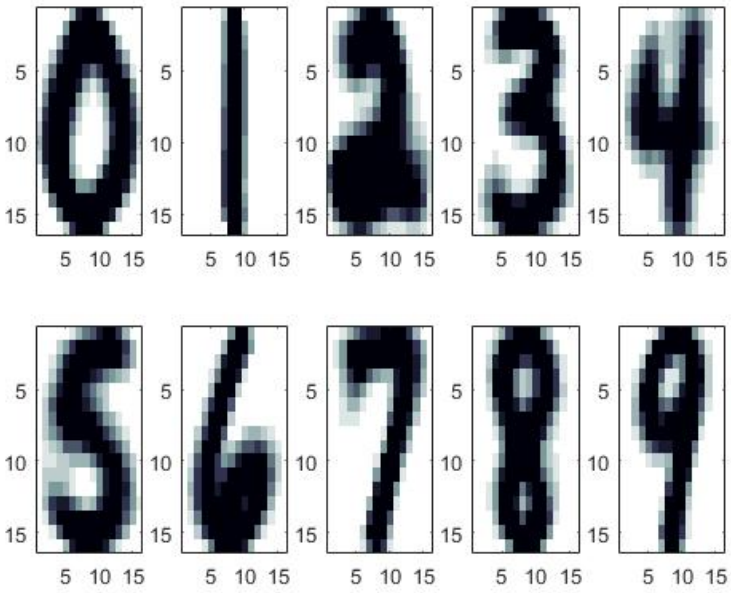


Figura 3: Immagini medie salvate nelle colonne di M.

Per vedere quanto è vicino il vettore *class* alla reale classificazione delle cifre di TestSet, lanciamo il seguente programma:

```
function precision_test(class, DigitsTest)
cont=0;
for i=1:2007
    if class(i)==DigitsTest(i)
        cont=cont+1;
    end
end
precisione=(cont/2007)*100
```

La sua esecuzione restituisce una precisione dell'80% circa.

```
>>precision_test(class, DigitsTest)

precisione=

      80.8670
```

ESERCIZIO 2

La funzione *svd_classifier* è:

```
function [class]=svd_classifier(TrainingSet, DigitsTraining,...
    TestSet, h)
% TrainingSet: m immagini vettorizzate su n componenti
% DigitsTraining: vettore di m componenti che
% riporta la cifra rappresentata
% di ogni immagine in TraininSet
% TestSet: matrice di immagini da classificare
% h :intero
% class: vettore contente la classificazione delle
% cifre nel TestSet

%contatore occorrenze di ogni cifra
cont=zeros(1,10);
for i=1:1707
    cont(DigitsTraining(i)+1)=cont(DigitsTraining(i)+1)+1;
end
%ordiniamo la matrice TrainingSet in base all'ordine di DigitsTest
[n,m]=size(TrainingSet);
M=TrainingSet';
M(:,n+1)=DigitsTraining;
B=sortrows(M,n+1);
B(:,end)=[];
M=B';
%calcolo dell'SVD degli A_i, plot degli U_i
%creazione matrice BigU
x=0;
y=cont(1);
BigU=zeros(256,10*h);
for i=1:10
    [U,S,V]=svd(M(:,x+1:y));
    if i<10
        x=y;
        y=y+cont(i+1);
    end
    subplot(2, 5, i)
    ima2(U(:,1))
    BigU(:,(i-1)*h+1:i*h)=U(:,1:h);
end

v=zeros(1,10);
for i=1:2007
    for k=1:10
        v(k)=norm(((eye(256)-(BigU(:,(k-1)*h+1:k*h))...
            *(BigU(:,(k-1)*h+1:k*h))')*TestSet(:,i)));
    end
    for k=1:10
        if v(k)==min(v)
            class(i)=k-1;
        end
```

```
    end
    v=zeros(1,10);
end
end
```

svd_classifier restituisce la seguente immagine:

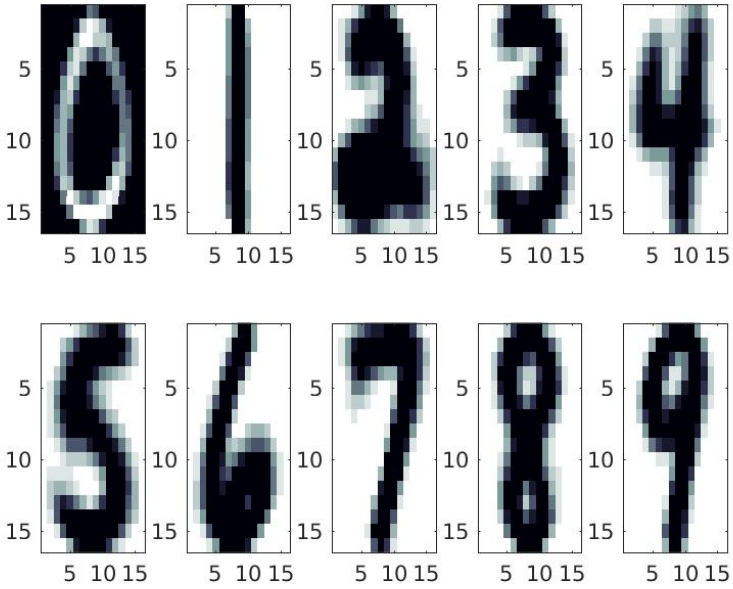


Figura 4: Immagini ottenute dal classificatore SVD.

La percentuale di immagini test salvate correttamente la calcoliamo con la function *percent_esatti*:

```
function percent_esatti(DigitsTest,DigitsTraining,TestSet,...
    TrainingSet)
percentuale=zeros(1,16);
a=1;
for h=5:20
    y=svd_classifier2(TrainingSet,DigitsTraining,TestSet,h)-...
        DigitsTest;
    esatti=2007-nnz(y);
    percentuale(a)=esatti/2007*100;
    a=a+1;
end
axis('equal')
plot([5:20],percentuale, '*b')
```

dove la funzione *svd_classifier2* è la funzione *svd_classifier* che non plotta i primi vettori singolari sinistri di U. L'immagine che si ottiene è:

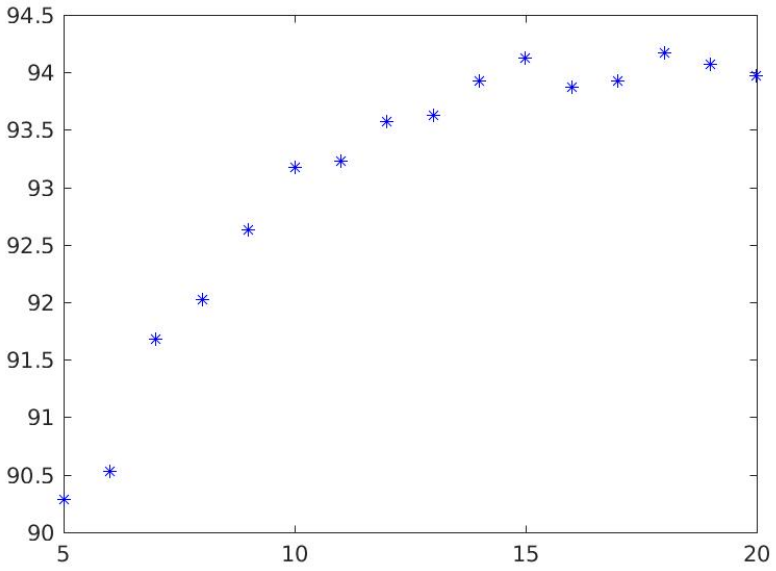


Figura 5: Percentuale di vettori classificati correttamente.

Per vedere se tutte le cifre vengono classificate con la stessa difficoltà lanciamo la function *percent_cifre*:

```
function percent_cifre(DigitsTest, DigitsTraining, TestSet,...
    TrainingSet)
%contatore occorrenze di ogni cifra
    cont=zeros(1,10);
    for i=1:2007
        cont(DigitsTest(i)+1)=cont(DigitsTest(i)+1)+1;
    end
%plot della percentuale di precisione per ogni cifra e per ogni
%h tra 5 e 20
for h=5:20
    percentuale=zeros(1,10);
    [class]=svd_classifier2(TrainingSet, DigitsTraining,...
        TestSet, h);
    for i=1:2007
        if class(i)==DigitsTest(i)
        percentuale(DigitsTest(i)+1)=...
            percentuale(DigitsTest(i)+1)+1;
        end
    end
    for k=1:10
        percentuale(k)=percentuale(k)/cont(k);
    end
    subplot(4,4,h-4)
    plot(0:9,percentuale,'*b')
    title(h)
end
```

che restituisce la seguente immagine:

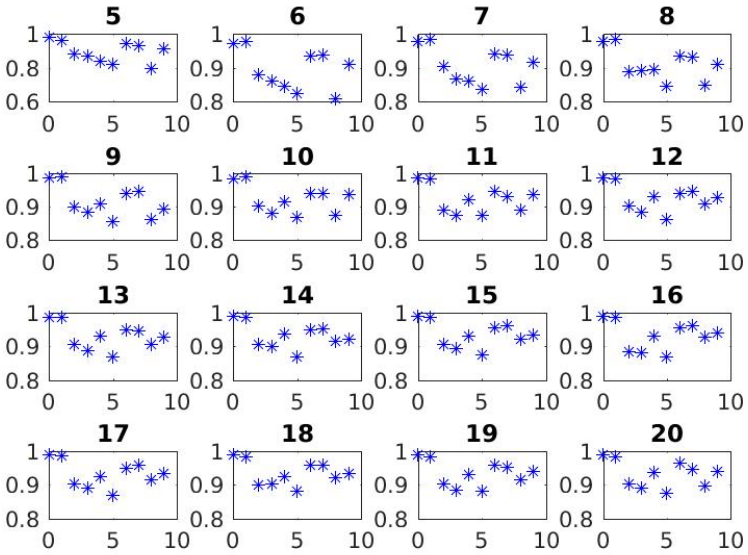


Figura 6: Percentuale di cifre esatte per ogni cifra e per ogni h tra 5 e 20.

Dall'immagine si evince che non tutti i numeri presentano la stessa difficoltà ad essere riconosciuti, infatti il 5 viene riconosciuto con percentuale molto minore dello 0.
Se ad esempio proviamo ad utilizzare un numero diverso di vettori singolari per rappresentare la cifra 5, osserviamo che la sua precisione cala. La seguente immagine mostra la precisione che si ottiene impostando a 4 il numero di vettori singolari del 5:

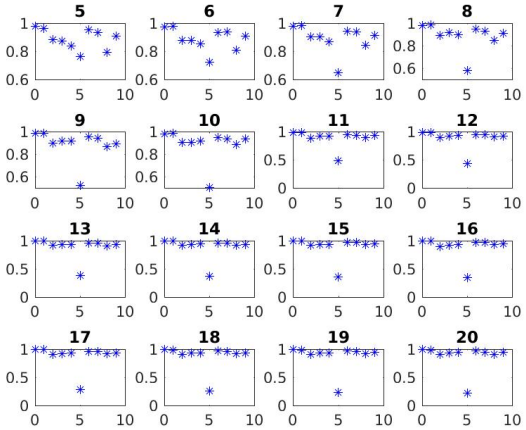


Figura 7: Numero di vettori singolari del 5=4.

TEST DELLE FUNZIONI

Testiamo le funzioni tramite la funzione *test_L3*:

```
>>testL3
Test frequenza passed
Test frequenza svd passed
Test frequenza svd passed
Test frequenza svd passed
Test frequenza svd passed
Test frequenza svd passed
```

3 LEZIONE IV, 10 MARZO 2017

ESERCIZIO 1

La funzione *cos_vector* è:

```
function vc=cos_vector(A, q)
% A: matrice termini-documenti
% q: vettore di query
% vc: vettore, vc(j) memorizza il coseno dell'angolo
% tra la colonna a_j e il vettore q

[m,n]=size(A);
vc=zeros(1,n);
for i=1:n
    vc(i)=(A(:,i)'*q)/(norm(q)*norm(A(:,i)));
end
```

ESERCIZIO 2

La funzione *precision_recall* è:

```
function [P, R]=precision_recall(vc, query_index, epsilon, M)
% vc vettore dei coseni
% epsilon: threshold
% query_idx: indice di una query in Q_med
% M: matrice che associa ad ogni query_index la lista
% dei documenti rilevanti
% P: valore di Precision
% R: valore di Recall

[n,m]=size(vc);
pos=zeros(m);
Dt=0;
Dr=0;
Nr=sum(M(query_index,:));

for i=1:m
    if vc(i)>epsilon
        Dt=Dt+1;
        pos(i)=1;
    end
end
a=find(pos);
b=find(M(query_index,:));
```

```
inter=intersect(a,b);
Dr=length(inter);
P=Dr/Dt;
R=Dr/Nr;
Dr
Dt
Nr
end
```

Testiamo la funzione:

```
>>x=523677;
>>query_index=mod(x,30)+1;
>>vc=cos_vector(A_med, Q_med(:,query_index));
>>epsilon=0.01;
>>[P, R]=precision_recall(vc, query_index, epsilon, Med_rel)

P=

    0.1111

R=

    0.9744
```

Per ottenere il grafico delle medie di precision e recall lanciamo lo script *media_prec*:

```
function media_prec(A,Q,M)
% A,Q,M sono A_med, Q_med, Med_rel
PM=zeros(1,10); %vettori delle medie
RM=zeros(1,10);
k=1;
for j=0.01:0.01:0.1
    Pm=0;
    Rm=0;
    for i=1:30
        vc=cos_vector(A,Q(:,i));
        [P, R]=precision_recall(vc, i, j, M);
        Pm=Pm+P;
        Rm=Rm+R;
    end
    PM(k)=Pm/30; %calcolo della media
    RM(k)=Rm/30;
    k=k+1;
end
a=0.01:0.01:0.1;
plot(a,PM,'or')
hold on
plot(a,RM,'ob')
```

Da questa funzione si ottiene il seguente grafico:

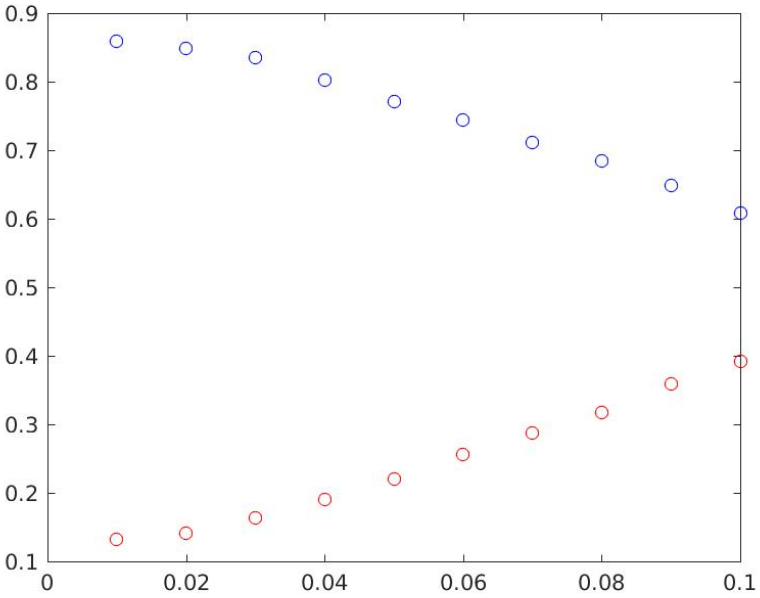


Figura 8: Medie di precision e recall, con precision rossa e recall blu.

ESERCIZIO 3

La funzione *LSI* è:

```
function [vc]=LSI(A, q, k)
% A: matrice terminixdocumenti
% q: vettore query
% k: intero
% vc: vettore dei coseni

[n,m]=size(A);
for j=1:m
    A(:,j)=A(:,j)/norm(A(:,j));
end
[U,S,V]=svds(A,k);
C=V*(S');
vc=zeros(1,k);
for j=1:k
    vc(j)=(C(j,:)*(U'*q))/(norm(C(j,:))*norm(U'*q));
end
```

Per controllare le medie con valori di k=25,50,75,100 usiamo lo script *media_LSI*:

```
function media_LSI(A,Q,M)
PM=zeros(1,4);
RM=zeros(1,4);
epsilon=0.01;
j=1;
```

```
for k=25:25:100
    Pm=0;
    Rm=0;
    for i=1:30
        vc=LSI(A, Q(:,i), k);
        [P, R]=precision_recall(vc, i, epsilon, M);
        Pm=Pm+P;
        Rm=Rm+R;
    end
    PM(j)=Pm/30;
    RM(j)=Rm/30;
    j=j+1;
end
x=25:25:100;
plot(x,PM,'or')
hold on
plot(x,RM,'ob')
```

Questa function ci restituisce il seguente grafico:

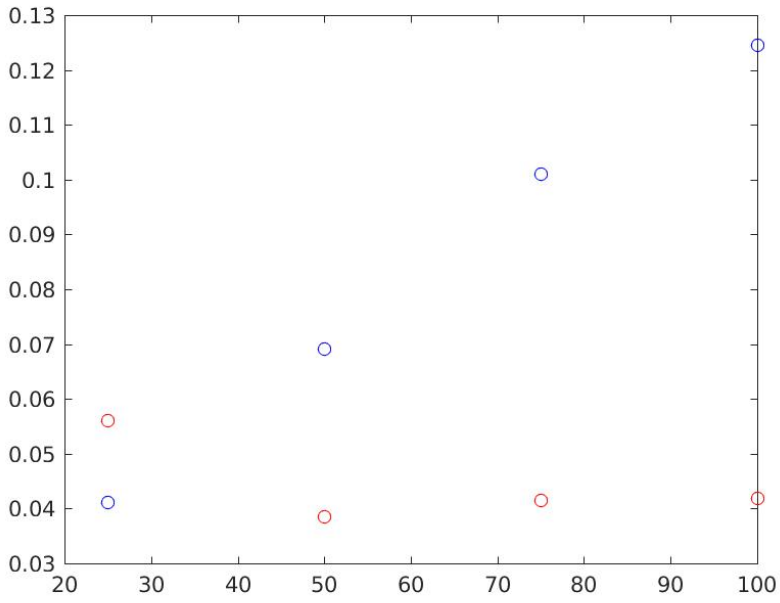


Figura 9: Medie di precision e recall, con precision rossa e recall blu.

4 LEZIONE V, 15 MARZO 2017

ESERCIZIO 1

La funzione *build_matrix* è:

```
function Phat=build_matrix(A, alpha)
% A: matrice di adiacenza di un grafo
% alpha: parametro di "teletrasporto" valore tipico 0.15
% Phat=(1-alpha)*Pbar+\alpha evT

[n,m]=size(A);
e=ones(n,1);
d=zeros(n,1);
v=e/n;
outdegree=A*ones(m, 1);
for i=1:n
    if outdegree(i)==0
        P(i,:)=A(i,:);
    else
        P(i, :)=A(i, :)/outdegree(i);
    end
end
for i=1:n
    if outdegree(i)==0
        d(i)=1;
    else
        d(i)=0;
    end
end
Pbar=P+d*v';
Phat=(1-alpha)*Pbar+alpha*e*v';
```

ESERCIZIO 2

La funzione *pagerank* è:

```
function [pr]=pagerank(A, alpha)
% A: matrice di adiacenza di un grafo
% alpha: parametro di teletrasporto valore tipico 0.15
% pr: vettore di pageRank costruito come l'autovettore sinistro
% di Phat relativo a lambda=1
Phat=build_matrix(A, alpha);
[V,D]=eigs(Phat',1);
pr=V';
```

Per fare il plot delle componenti vettore di PageRank usiamo lo script *pageplot*:

```
function pageplot(A)
hold on
alpha=[0.15, 0.1, 0.01, 10^-8, 10^-12];
plot(pagerank(A,alpha(1)),'y');
plot(pagerank(A,alpha(2)),'m');
plot(pagerank(A,alpha(3)),'c');
plot(pagerank(A,alpha(4)),'r');
plot(pagerank(A,alpha(5)),'g');
```

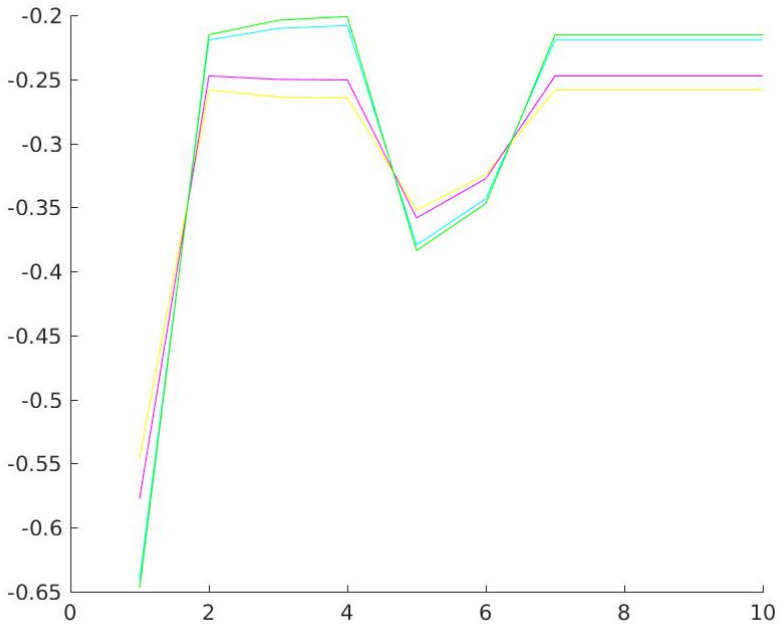


Figura 10: Grafico componenti PageRank.

ESERCIZIO 3

La funzione *P_times_x* è:

```
function [y]=P_times_x(P, x, alpha)
% P: matrice ottenuta dalla matrice di adiacenza
% scalando le righe per l'outdegree
% alpha: parametro di teletrasporto
% x: vettore
[m,n]=size(P);
e=ones(n,1);
v=e/n;
y=(1-alpha)*P'*x;
gamma = 1-norm(y,1);
y=y+gamma*v;
end
```

ESERCIZIO 4

Parte 1

La funzione *pr_powermethod* è:

```
function [pr]=pr_powermethod(A, alpha)
% A: matrice di adiacenza di un grafo
% alpha: parametro di teletrasporto
% pr: vettore di PageRank
[n,m]=size(A);
somma = sum(A, 2);
B = find(somma==0);
somma(B) = 1;
C = spdiags(1./somma, 0, n, n);
D = C*A;

i = 1;
pr = rand(n, 1);
pr = pr/norm(pr, 1);
p = zeros(n, 1);
while((i<100) && (norm(pr-p)>10^(-7)))
    p = pr;
    pr = P_times_x(D, pr, alpha);
    error(i) = norm(pr-p);
    i = i+1;
end

plot(error,'b')

end
```

Questa funzione traccia anche il grafico degli errori. Usandola sulla matrice *stanford_web* otteniamo il seguente grafico:

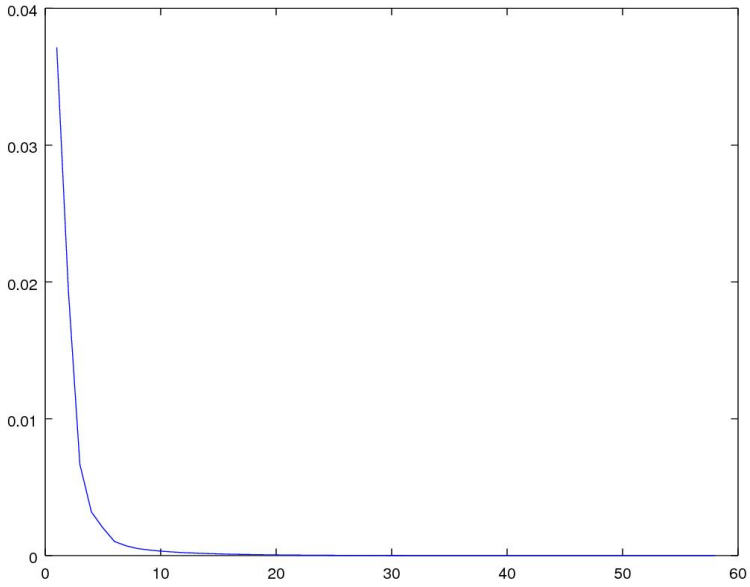


Figura 11: Grafico errori di pr_powermethod.

Parte 2

Creiamo la funzione *P_times_x_norm*:

```
function y = P_times_x_norm(A, x, alpha)

y = norm(x, 1)*P_times_x(A, x/norm(x, 1), alpha);

end
```

La funzione *pagerank_eigs* è:

```
function [pr]=pagerank_eigs(A,alpha)

[n,m]=size(A);
somma = sum(A, 2);
B = find(somma==0);
somma(B) = 1;
C = spdiags(1./somma, 0, n, n);
D = C*A;

y=@(x) P_times_x_norm(D, x, alpha);
[V,D]=eigs(y, n, 1);
pr=V;
```

Parte 3

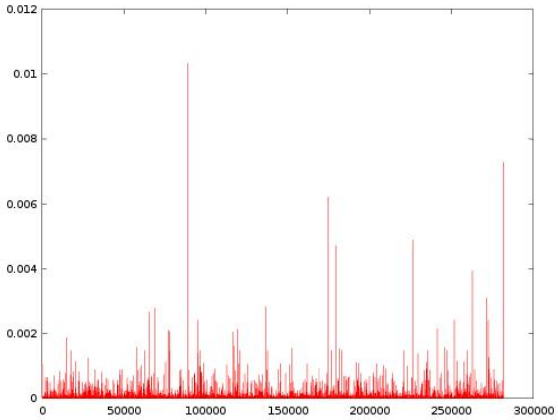
Usiamo la funzione *pr_powermethod_2* che si ottiene dalla funzione *pr_powermethod* semplicemente cancellando l'istruzione di *plot*. Per confrontare la velocità di convergenza dei due metodi usiamo lo script *confronto_tempi*:

```
function confronto_tempi(A)

disp('powermethod con alpha=0.15')
tic;
a=pr_powermethod_2(A,0.15);
toc;

disp('pagerank_eigs con alpha=0.15')
tic;
b=pagerank_eigs(A,0.15);
toc;

figure(1);
plot(a - b, 'b');

disp('powermethod con alpha=0.01')
tic;
c=pr_powermethod_2(A,0.01);
toc;
plot(c, 'r')

disp('pagerank_eigs con alpha=0.01')
tic;
d=pagerank_eigs(A,0.01);
toc;

figure(2);
plot(c - d, 'b');

end
```

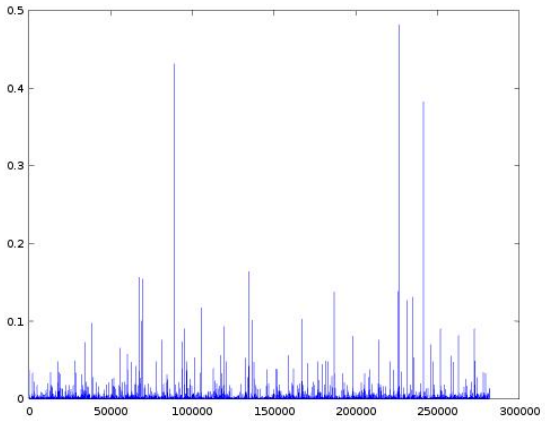
Eseguendo la funzione *confronto_tempi* si ottiene il seguente output:

```
>> confronto_tempi(A)
powermethod con alpha=0.15
Elapsed time is 10.497579 seconds.
pagerank_eigs con alpha=0.15
Elapsed time is 11.084786 seconds.
powermethod con alpha=0.01
Elapsed time is 16.675193 seconds.
pagerank_eigs con alpha=0.01
Elapsed time is 14.549444 seconds.
```

e le seguenti 2 immagini:



(a) *alpha=0.15*



(b) *alpha=0.11*

Figura 12: Immagini ottenute da *confronto_tempi*.

Da queste immagini si può vedere come le componenti dei vettori differenza siano tutte molto piccole, inferiori in valore assoluto a 0.012 nel caso *alpha = 0.15* ed inferiori a 0.5 nel caso *alpha = 0.11*. Le soluzioni ottenute con i due metodi sono dunque molto vicine tra loro.

5 LEZIONE VI, 17 MARZO 2017

ESERCIZIO 1

La funzione *Hiterate_HITS* è:

```
function [h, a]=Iterate_HITS(A)
% A matrice di adiacenza di un grafo
% h e a punteggi di Hub e autority

iter=3000;
epsilon=10^-16;

h0=ones(size(A,2),1);
a0=ones(size(A,2),1);
h=A*a0;
a=a0;
B=A';
i=0;
while (norm(h-h0)>epsilon || norm(a-a0)>epsilon) & i<iter
    h0=h;
    a0=a;
    a=B* h0;
    h=A*a;
    h=h/norm(h, 2);
    a=a/norm(a, 2);
    i=i+1;
end
```

Confontiamo i risultati con quelli ottenuti tramite il comando *eigs*:

```
>>load('QMATRIX.mat')
>>[h,a]=Iterate_HITS(Q)
h =

   0.00000
   0.00000
   0.52573
   0.00000
   0.00000
   0.00000
   0.42533
   0.42533
   0.42533
   0.42533


a =
```

```
   0.97325
   0.00000
   0.00000
   0.22975
   0.00000
   0.00000
   0.00000
   0.00000
   0.00000
   0.00000
>> [A,~]=eigs(Q'*Q,1,'LM')

A =

   -0.9732
    0.0000
    0.0000
   -0.2298
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000

>> [H,~]=eigs(Q*Q',1,'LM')

H =

    0.0000
   -0.0000
   -0.5257
   -0.0000
    0.0000
   -0.0000
   -0.4253
   -0.4253
   -0.4253
   -0.4253
```

ESERCIZIO 2

La funzione *rank_compare* è:

```
function [top_idx, pr_idx, auth_idx]=rank_compare(A, n)}
% A matrice di adiacenza di un grafo
% n: intero positivo
% top_idx: vettore di indici dei primi n nodi ordinati in base
% a somma di indegree e outdegree
% pr_idx: vettore di indici dei primi n nodi ordinati
% in base a pagerank
% auth_idx:vettore di indici dei primi n nodi ordinati in base
% al punteggio di authority

indegree = sum(A);
outdegree = sum(A');
sumdegree = indegree + outdegree;
[B,I]=sort(sumdegree,'descend'); %ordino il vettore sumdegree
top_idx=I(1:n);
C=A(top_idx,top_idx); %riordino a secondo
                      %le prime n componenti di I
pr=pagerank(C,0.15);
[Bpr,Ipr]=sort(pr,'descend');
pr_idx=top_idx(Ipr);
[a,h]=Iterate_HITS(C);
[Bh,Ih]=sort(h,'descend');
auth_idx=top_idx(Ih);
```

Mostriamo le prime dieci componenti dei 3 vettori di ranking:

```
>>A=spconvert(stanford_web);
>>[top_idx, pr_idx, auth_idx]=rank_compare(A,10);
>>B=[top_idx; pr_idx; auth_idx]'

B=

   226411   241454    81435
   234704   245659   214128
   105607   167295   198090
   241454   234704   105607
   167295   226411   226411
    38342    38342   234704
    81435    81435   167295
   214128   214128    38342
   198090   198090   245659
   245659   105607   241454
```

ESERCIZIO 3

Per confrontare l'ordinamento indotto da ac con quelli ottenuti con le altre strategie, modifichiamo leggermente la function *rank_compare* in

modo che ci restituisca la matrice ordinata e troncata; per fare questo basta aggiungere nella dichiarazione di funzione, tra le variabili di output, la variabile C:

```
function [top_idx, pr_idx, auth_idx,C]=rank_compare(A, n)
```

Creiamo quindi la function *centrality.m*:

```
function [hc, ac]=centrality(A)
% A : matrice di adiacenza
% hc: misura di centralita' come hub
% ac: misura di centralita' come authority

[n,m]=size(A);
B=[zeros(n,n), A; A', zeros(n,n)];
E=expm(B);
hc=diag(E(1:n,1:n));
ac=diag(E(n+1:2*n,n+1:2*n));
```

Calcoliamo la matrice C e i vari ordinamenti di rank_compare con il comando:

```
>>A=spconvert(stanford_web);
>>[top_idx,pr_idx,auth_idx,C]=rank_compare(A,1000);
```

Procediamo calcolando ac con centrality, calcolandone inoltre l'ordinamento

```
>>[ac,~]=centrality(C);
>>[~,ordac]=sort(ac,'descend');
```

Quindi facciamo il plot di questi 4 vettori:

```
>>figure(1)
>>plot(ordac)
>>figure(2)
>>plot(top_idx)
>>figure(3)
>>plot(pr_idx)
>>figure(4)
>>plot(auth_idx)
```

Ottenendo quindi le seguenti immagini:

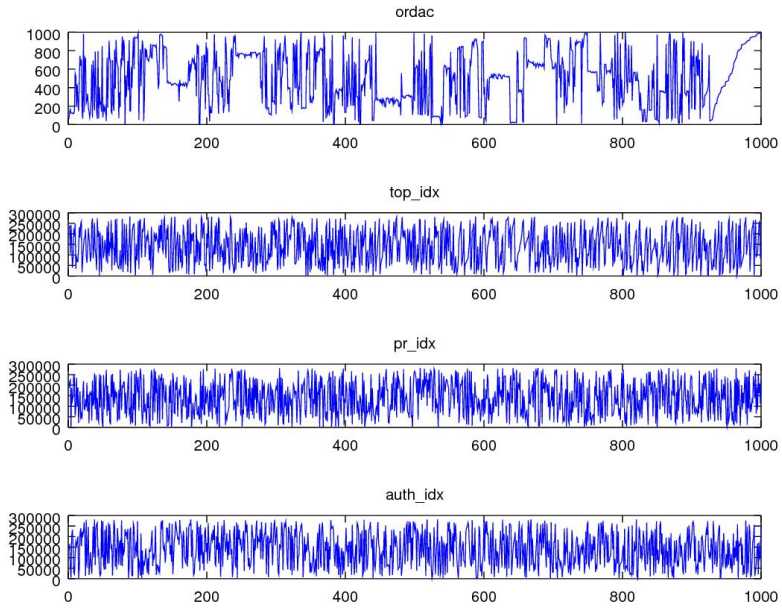


Figura 13: Immagini degli ordinamenti ac, top, pr, auth.

6 LEZIONE VII, 22 MARZO 2017

ESERCIZIO 1

La funzione *similarity* è:

```
function [S_user, S_items]=similarity(A)
% A; matrice dei rating utenti-items
% S_user: matrice di similarita' (misura del coseno)
% tra ogni coppia di utenti di A
% S_items; matrice di similarita' (misura del coseno)
%tra ogni coppia di items di A

[m,n]=size(A);
B=A;
S_user=zeros(m,m);
S_items=zeros(n,n);
%costruiamo la matrice S_user
for p=1:m
    B(p,:)=B(p,:)/norm(B(p,:));
end
for i=1:m
    for j=1:m
        S_user(i,j)=B(i,:)*B(j,:)';
    end
end
%costruiamo la matrice S_items
B=A;
for q=1:n
    B(:,q)=B(:,q)/norm(B(:,q));
end
for i=1:n
    for j=1:n
        S_items(i,j)=C(:,i)'*C(:,j);
    end
end
```

ESERCIZIO 2

La funzione *recommend* è:

```
function [items_list]=recommend(A, user)
% A: matrice dei rating utenti-items
% user: indice di un utente
% items_list: insieme di indici di item da raccomandare a user

[m,n]=size(A);
[S_user,~]=similarity(A);
items_list=zeros(1,n);
[~,I]=sort(S_user(user,:),'descend');
i_max=I(2);
for j=1:n
    if ((A(user,j)==0) & ((A(i_max,j)==4) || (A(i_max,j)==5)))
        items_list(j)=1;
    end
end
```

ESERCIZIO 3

La funzione *accuracy* è:

```
function [p, r]=accuracy(B, item_list, user)
% B: matrice user-items (test-set)
% item_list: insieme di indici
% user: identificativo di un utente
% p: valore di precision
% r: valore di relall

[n,m]=size(B);
tp=0;
fp=0;
fn=0;
for j=1:m
    if (items_list(j)>0) & (B(user,j)>=4)
        tp=tp+1;
    elseif (items_list(j)>0) & (0<B(user,j)<4)
        fp=fp+1;
    elseif (items_list(j)==0) & (B(user,j)>=4)
        fn=fn+1;
    end
end
p=tp/(tp+fp);
r=tp/(tp+fn);
```

Per calcolare l'insieme di item da raccomandare per ogni utente nel Test-set usiamo la funzione *accuracy_2*:

```
function [p,r,p_media,r_media]=accuracy_2(A,B)
j=0;
[m,~]=size(B);
for i=1:m
    if norm(B(i,:))~=0
        j=j+1;
        items_list=recommend(A,i);
        [p(j),r(j)]=accuracy(B,items_list,i);
    end
end

p_media=sum(p)/j;
r_media=sum(r)/j;
```

ESERCIZIO 4

Parte 1

La funzione *transform* è:

```
function [A_mean]=transform(A)}
% A: matrice user-items
% A_mean: marice user_items

[m,n]=size(A);
A_mean=A;
for i=1:m
    mean=mean(A(i,:));
    for j=1:n
        if A_mean(i,j)==0
            A_mean(i,j)=mean;
        end
    end
end
```

Parte 2

La funzione *pure_svd* è:

```
function [C]=pure_svd(A, k)}
% A: matrice user-items
% C: matrice user_items

[m,n]=size(A);
A_mean=transform(A);
X=A;
C=zeros(m,n);
%SVD di rango k di A_mean
[S,V,D]=svds(A_mean,k);
M=S*V*D';
for i=1:m
    for j=1:n
        if X(i,j)==0
            X(i,j)=M(i,j);
        end
        if X(i,j)<=0
            C(i,j)=0;
        elseif 0<X(i,j)<5
            C(i,j)=round(X(i,j));
        else
            C(i,j)=5;
        end
    end
end
```

Parte 3

La funzione *recommend_svd* è:

```
function [U]=recommend_svd(A,C)
% A: matrice user-item
% C: matrice user-item ottenuta con la funzione pure_svd
% U: matrice user-item binaria

[m,n]=size(A);
U=zeros(m,n);
for i=1:m
    for j=1:n
        if (A(i,j)==0) & (C(i,j)>=4)
            U(i,j)=1;
        end
    end
end
```

Per calcolare i valori medi di precision e recall su tutti gli utenti del Test-set usiamo la seguente funzione:

```
function [p,r,p_media,r_media]=accuracy_3(A,B,k)
j=0;
[n,m]=size(B);
C=pure_svd(A,k);
U=recommend_svd(A,C);
for i=1:n
    if norm(B(i,:))~=0
        j=j+1;
        [p(j),r(j)]=accuracy(B,U(i,:),i);
    end
end
p_media=sum(p)/j;
r_media=sum(r)/j;
```