



Mini-progetto

Decomposizione CUR su matrici sparse: un'implementazione

Elena Caruso

2022/2023

Indice della presentazione

- 1. Obiettivi** che si vogliono raggiungere
- 2. Metodi utilizzati**
- 3. Esperimenti**
- 4. Conclusioni**

Nel nostro Progetto ci siamo occupate di approfondire ed implementare su python la decomposizione CUR.

Data una matrice $A_{m \times n}$ si ha che:

$$A_{CUR} = C_{m \times k} U_{k \times k} R_{k \times n}$$

- k è un numero fissato
- Le matrici C e R sono costituite da k colonne e k righe scelte casualmente da A e riscalate
- Per fare in modo che l'approssimazione sia migliore possibile le righe e le colonne vengono scelte con probabilità diverse che dipendono dalla norma di Frobenius

Obiettivi

- La CUR non è molto nota, appare per la prima volta in un articolo di Penrose (1956), mentre studi più recenti ne propongono alcuni algoritmi per il calcolo come Drineas et.al. (2006)
- La CUR è utile quando si hanno matrici molto grandi e sparse perchè preserva la struttura originale della matrice di partenza e ciò permette di memorizzare più facilmente le matrici fattorizzanti e di essere computazionalmente più efficiente.
- Alcuni contesti in cui è stata applicata in modo efficace sono la bioinformatica, i sistemi di raccomandazione e la compressione di immagini

Gli aspetti che abbiamo approfondito riguardano l'efficienza e l'efficacia della CUR

- al variare della quantità di dati
- al variare del parametro k

“There is a theory that guarantees convergence to M as r gets larger, but typically you have to make r so large to get, say within 1% that the method becomes impractical. Nevertheless, a decomposition with a relatively small value of r has a good probability of being a useful and accurate decomposition.”

(Leskovec et al., “Mining of Massive Datasets”, 2010)

Abbiamo confrontato la CUR con la SVD che sappiamo essere la migliore decomposizione a rango basso in merito a:

- Accuratezza dell'approssimazione
- Tempo impiegato
- Spazio occupato

1) Accuratezza

L'accuratezza è stata valutata calcolando la norma di Frobenius per la CUR e per la SVD

$$||A - \hat{A}||_F = \sqrt{\sum_{i,j} (a_{ij} - \hat{a}_{ij})^2}$$

E ne è stato studiato il comportamento

- Al variare del parametro k in un intervallo di valori che va da 50 a 450, mantenendo costante la quantità di dati pari a 2000 film
- Al variare della quantità di dati da 500 a 3150 mantenendo costante k

1) Tempo e spazio

Per questa valutazione abbiamo

- Scelto diversi valori del parametro k da 50 a 950 mantenendo costante la dimensione dei dati pari a 8000 film (circa 45000000 entrate)
- Selezionato diverse quantità di dati tra 500 a 3150 film mantenendo costante il valore di k

Metodi e Metodologia utilizzati

Metodi utilizzati

1. Strumenti utilizzati (librerie e device)
2. Algoritmi principali:
 - a. Algoritmo di archiviazione dei dati
 - b. Implementazione di CUR
 - c. Calcolare l'accuratezza dell'approssimazione del metodo CUR
 - d. Metodi in breve per calcolare la SVD
3. Metodologia generale utilizzata per effettuare gli esperimenti

1.

Strumenti utilizzati

❑ Per la creazione di dataset:

- `Sqlite3`

[\[https://www.sqlite.org/index.html\]](https://www.sqlite.org/index.html)

❑ Per calcoli della SVD:

- `numpy`

[\[https://numpy.org\]](https://numpy.org)

- `scipy.sparse.linalg`

[\[https://docs.scipy.org/doc/scipy/reference/sparse.linalg.html\]](https://docs.scipy.org/doc/scipy/reference/sparse.linalg.html)

❑ Per la gestione di matrici sparse:

- `scipy.sparse.coo_matrix`

[\[https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.coo_matrix.html\]](https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.coo_matrix.html)

❑ Per valutazioni del tempo e dello spazio occupato:

- `psutil`

[\[https://psutil.readthedocs.io/en/latest/\]](https://psutil.readthedocs.io/en/latest/)

- `codetiming`

[\[https://pypi.org/project/codetiming/\]](https://pypi.org/project/codetiming/)

1.

Strumenti per la sperimentazione

Device utilizzati.

- ❑ Per la parte sperimentale sono stati usati computer con caratteristiche differenti:

Sistema Operativo	Processore	RAM
Windows 11	11th Gen Intel(R) Core(TM) i7	16 GB
Windows 10	Intel(R) Core(TM) i5	8 GB

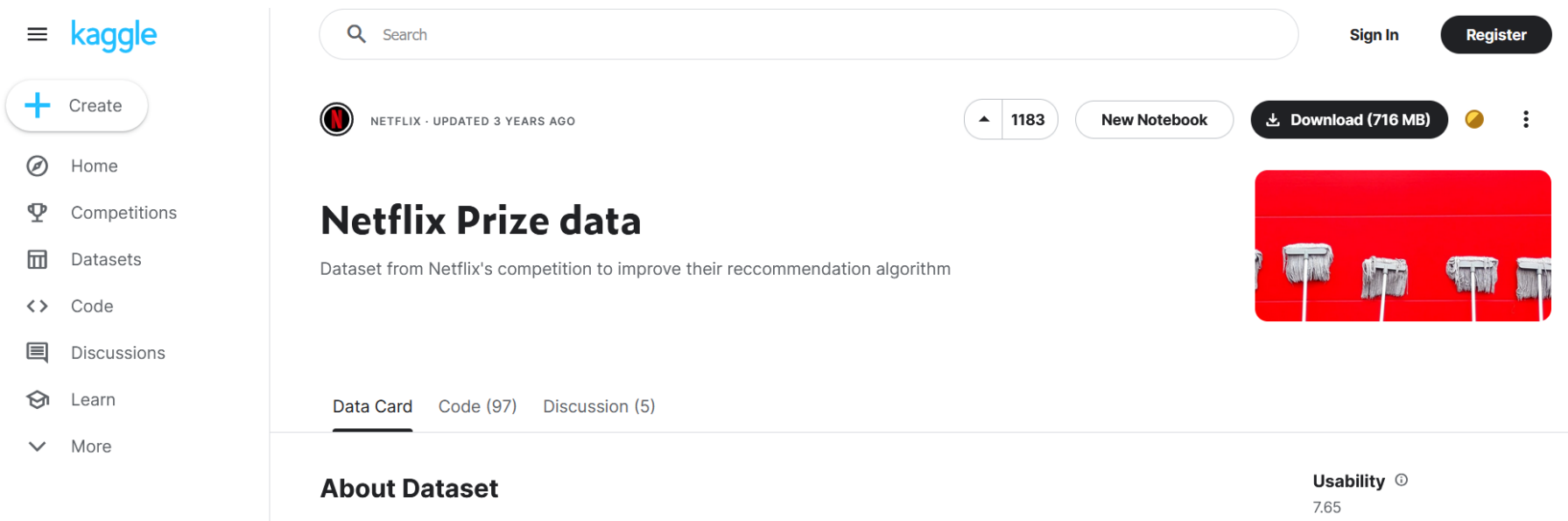
2. a.

Dataset

Il dataset che utilizziamo è il dataset **Netflix Prize Data**¹, è costituito da:

- Circa **cento milioni di valutazioni** espresse su una **scala da 1 a 5**
- di **480.189** utenti
- su **17.770** film.

I dati in totale hanno una mole di **2GB** e sono suddivisi in chunk da 500MB.



The screenshot shows the Kaggle website interface. On the left is a sidebar with navigation links: Home, Competitions, Datasets, Code, Discussions, Learn, and More. The main content area displays the 'Netflix Prize data' dataset page. At the top, there's a search bar and buttons for 'Sign In' and 'Register'. Below the search bar, the dataset is identified as 'NETFLIX - UPDATED 3 YEARS AGO' with 1183 votes. There are buttons for 'New Notebook' and 'Download (716 MB)'. The dataset title 'Netflix Prize data' is prominently displayed, followed by a description: 'Dataset from Netflix's competition to improve their recommendation algorithm'. Below this, there are tabs for 'Data Card' (selected), 'Code (97)', and 'Discussion (5)'. At the bottom, the 'About Dataset' section is visible, showing a 'Usability' score of 7.65.

1. <https://www.kaggle.com/datasets/netflix-inc/netflix-prize-data>

2. a.

Dataset



The screenshot shows a Windows Notepad window titled "combined_data_1 - Blocco note". The window contains a list of data points, each consisting of a numerical ID, a comma, a small integer, another comma, and a date in YYYY-MM-DD format. The data points are as follows:

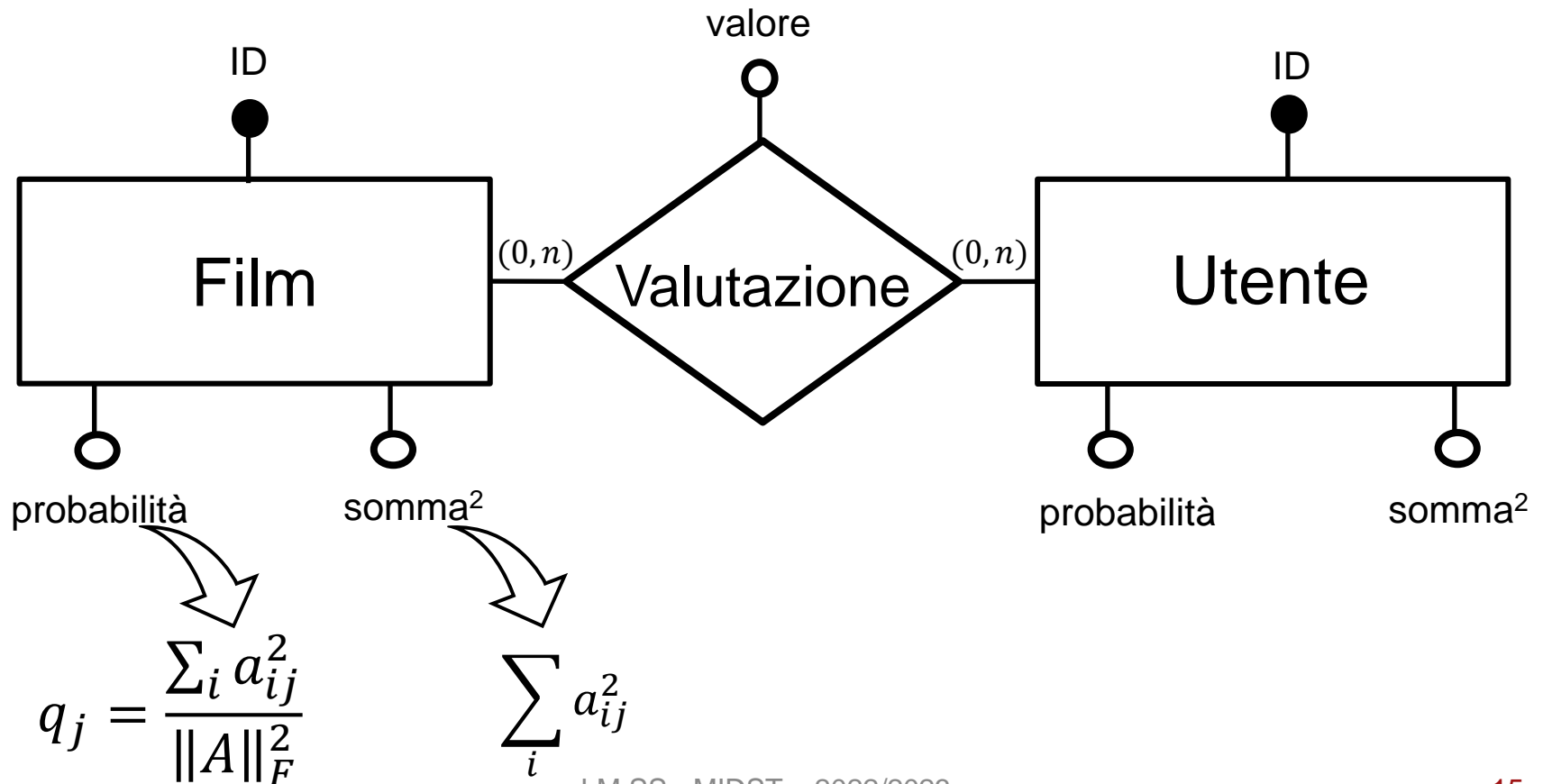
ID	Value	Date
1488844	3	2005-09-06
822109	5	2005-05-13
885013	4	2005-10-19
30878	4	2005-12-26
823519	3	2004-05-03
893988	3	2005-11-17
124105	4	2004-08-05
1248029	3	2004-04-22
1842128	4	2004-05-09
2238063	3	2005-05-11
1503895	4	2005-05-19
2207774	5	2005-06-06
2590061	3	2004-08-12

The status bar at the bottom of the window indicates "Linea 1, colonna 100%", "Unix (LF)", and "UTF-8".

- I dati sono divisi in 4 file .txt

2. a.

Schema SQL



2. a.

Archiviazione dei dati

1. Viene **letto il file** riga per riga e i dati vengono inseriti nella tabella `valutazione` con record di tipo *(film, utente, valutazione)*.
2. Vengono **create e riempite le tabelle** `utente` e `film`.
3. Vengono **creati i dizionari** che servono a **dare nuovi indici agli utenti e ai film**, in modo tale da poter costruire tutte le matrici in formato `coo_matrix`.

indiceU{id utente: nuovo indice}
indiceF{id film: nuovo indice}

dizionarioUte{id utente: probabilità}
dizionarioFilm{id film: probabilità}

2. b.

Calcolare C, U e R

1. E' necessario scegliere gli k film e utenti.
 2. Poi si calcolano le tre matrici separatamente.
- ❑ Il metodo trovato per calcolare C,U e R ha ordine di complessità pari a $o(kn + k^3)$, dove n è il numero di entrate e k è il numero di film e utenti **scelti**.
 - ❑ Il codice rende possibile decidere di avere le matrici sia in formato di `coo_matrix` sia in una tabella SQL.

2. b.

La scelta

dizionarioUte{id utente: probabilità}
dizionarioFilm{id film: probabilità}

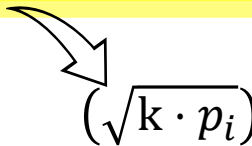
```
#A dal numero k di elementi da scegliere e dai dizionari,  
#ritorna k utenti e k film che vengono scelti casualmente  
#seguendo una distribuzione multinomiale  
def sceltak(k,dizionarioUte,dizionarioFilm):  
    #A partire dai dizionari crea 4 np.array, necessari per la scelta casuale  
    ArUte=np.array(list(dizionarioUte.keys()))  
    ArPer=np.array(list(dizionarioUte.values()))  
    AcFil=np.array(list(dizionarioFilm.keys()))  
    AcPer=np.array(list(dizionarioFilm.values()))  
  
    #vengono scelti k utenti e k film seguendo una distribuzione multinomiale.  
    #Le probabilità sono dati dagli oggetti dei dizionari  
    kutenti=np.random.choice(ArUte, size=k, replace=True, p=ArPer)  
    kfilm=np.random.choice(AcFil, size=k, replace=True, p=AcPer)  
  
    print("Effettuata la scelta dei",k,"elementi.")  
    return kutenti,kfilm
```

2. b.

Calcolo di C

```
#Dando il numero k di elementi necessari, il dizionario riferito a film e il vettore dei k film scelti,
#viene calcolata la matrice C in SQL, se svd è True viene ritornata C anche in matrice.
def calcC(k, curs, con, dizionarioFilm, kfilm, svd=False):
    curs.execute("CREATE TABLE IF NOT EXISTS c(utenteI INT, filmI INT, val REAL)")
    #il valore di valutazione viene diviso per sqrt(dizionarioFilm.get(kfilm[i])*k)
    for i in range(k):
        curs.execute('INSERT INTO c SELECT indice, ?, CAST(val AS REAL)/? FROM indiceU
            NATURAL JOIN (SELECT val, utente FROM valutazione WHERE film=?)',
            i, math.sqrt(dizionarioFilm.get(kfilm[i])*k), int(kfilm[i]))
    curs.execute("CREATE INDEX cI ON c (utenteI, filmI)")
    con.commit()
    print("Calcolata C, è contenuta nella tabella sql c.")
    if svd:
        indUteC=[]
        indFilmC=[]
        valC=[]

        curs.execute("SELECT * FROM c")
        cc=curs.fetchall()
        for c in cc:
            indUteC.append(c[0])
            indFilmC.append(c[1])
            valC.append(float(c[2]))
        C=coo_matrix((valC, (indUteC, indFilmC)), shape=(mm, k))
    return C
```



$$(\sqrt{k \cdot p_i})$$

2. b.

L'algoritmo

```
#Funzione che a partire dai dizionari Utente e Film calcola la CUR per il k dato.  
#Ritorna il tempo trascorso  
def calcolaCUR(k, curs, con, dizionarioUte, dizionarioFilm):  
  
    t = Timer(name="class")  
    t.start()  
    kutenti, kfilm=sceltak(k, dizionarioUte, dizionarioFilm)  
    calcC(k, curs, con, dizionarioFilm, kfilm)  
    calcR(k, curs, con, dizionarioUte, kutenti)  
    calcU(k, curs, con, kutenti, kfilm)  
    fine=t.stop()  
  
    return fine
```

2. c.

Calcolo dell'accuratezza di CUR

- ❑ Una delle principali difficoltà del progetto è stata **calcolare l'accuratezza** per la CUR in modo tale da valutare anche casi con **matrici molto grandi**.
- ❑ Per valutare l'accuratezza usiamo la norma di Frobenius:

$$||A - CUR||_F = \sqrt{\sum_{i,j} (a_{ij} - CUR_{ij})^2},$$

è necessario moltiplicare le tre matrici C, U e R.

- ❑ Dopo vari tentativi, usando sia metodi su SQL che su Python, abbiamo deciso di calcolare il prodotto con un metodo del tipo ***dividi et impera***.



2. c.

Calcolo dell'accuratezza di CUR

- La moltiplicazione di C,U e R, diventa

$$\begin{pmatrix} CUR_1 & CUR_2 \\ CUR_3 & CUR_4 \end{pmatrix} = \begin{pmatrix} C_1 & C_2 \\ C_3 & C_4 \end{pmatrix} * \begin{pmatrix} U_1 & U_2 \\ U_3 & U_4 \end{pmatrix} * \begin{pmatrix} R_1 & R_2 \\ R_3 & R_4 \end{pmatrix} =$$

$$\begin{pmatrix} C_1U_1R_1 + C_2U_3R_1 + C_1U_2R_3 + C_2U_4R_3 & C_1U_1R_2 + C_2U_3R_2 + C_1U_2R_4 + C_2U_4R_4 \\ C_3U_1R_1 + C_4U_3R_1 + C_3U_2R_3 + C_4U_4R_3 & C_3U_1R_2 + C_4U_3R_2 + C_3U_2R_4 + C_4U_4R_4 \end{pmatrix},$$

- Per calcolare la distanza viene utilizzata, la norma di Frobenius:

$$\|A - X\|_F = \sqrt{\sum_{i,j} (a_{ij} - x_{ij})^2},$$

che viene calcolata per ogni quadrante della matrice per poi sommarla insieme, si ottiene quindi:

$$\|A - CUR\|_F \leq \|A_1 - CUR_1\|_F + \|A_2 - CUR_2\|_F + \|A_3 - CUR_3\|_F + \|A_4 - CUR_4\|_F .$$

- Nel codice per calcolare la norma di Frobenius useremo il comando:

```
scipy.sparse.linalg.norm()
```

2. c.

Calcolo dell'accuratezza di CUR

```
def prodottoCURpy(curs,A1,A2,A3,A4,k,uMAX,fMAX):
    k=int(k)
    C1,C2=trovaC1C2(curs,k,uMAX)
    U1,U2,U3,U4=trovaU1_4(curs,k)
    R1,R3=trovaR1R3(curs,k,fMAX)
    #Calcoliamo CUR1 e la distanza di frob relativa
    CUR1=coo_matrix(C1.dot(U1).dot(R1)+C2.dot(U3).dot(R1)+C1.dot(U2).dot(R3)+C2.dot(U4).dot(R3))
    diff=norm(CUR1-A1,'fro')
    del CUR1
    #Calcoliamo CUR2 e la distanza di frob relativa
    R2,R4=trovaR2R4(curs,k,fMAX)
    CUR2=coo_matrix(C1.dot(U1).dot(R2)+C2.dot(U3).dot(R2)+C1.dot(U2).dot(R4)+C2.dot(U4).dot(R4))
    del C1,C2
    diff=diff+norm(CUR2-A2,'fro')
    del CUR2
    #Calcoliamo CUR3 e la distanza di frob relativa
    C3,C4=trovaC3C4(curs,k,uMAX)
    CUR3=coo_matrix(C3.dot(U1).dot(R1)+C4.dot(U3).dot(R1)+C3.dot(U2).dot(R3)+C4.dot(U4).dot(R3))
    del R1,R3
    diff=diff+norm(CUR3-A3,'fro')
    del CUR3
    #Calcoliamo CUR4 e la distanza di frob relativa
    CUR4=coo_matrix(C3.dot(U1).dot(R2)+C4.dot(U3).dot(R2)+C3.dot(U2).dot(R4)+C4.dot(U4).dot(R4))
    del C3,C4,R2,R4
    diff=diff+norm(A4-CUR4,'fro')
    del CUR4
    print("Fatto il prodotto.")
    return diff
```



2. d.

Strumenti per la SVD

- ❑ Per effettuare la SVD decidiamo di usare due funzioni già implementate:
 - `numpy.linalg.svd()`
 - `scipy.sparse.linalg.svds()`
- ❑ La prima calcola la SVD per matrici **dense**, la seconda per matrici **sparse**.
- ❑ E' necessario implementare un nuovo metodo per leggere il file in modo tale da avere la matrice A in memoria centrale.

3.

Metodologia utilizzata

- ❑ Per arrivare agli obiettivi richiesti abbiamo deciso di effettuare per ognuno dei tre punti **due diversi esperimenti**:
 1. Uno doveva valutare l'**accuratezza**
 2. Il secondo doveva valutare il **tempo** e lo **spazio**.
- ❑ Il motivo è dato dal fatto che il calcolo dell'accuratezza comporta moltiplicare matrici molto grandi, per cui serve del tempo e molto spazio.
- ❑ Questo avrebbe rallentato il processo e non ci avrebbe dato modo di utilizzare tutti i dati possibili.

Risultati: CUR tempo e spazio

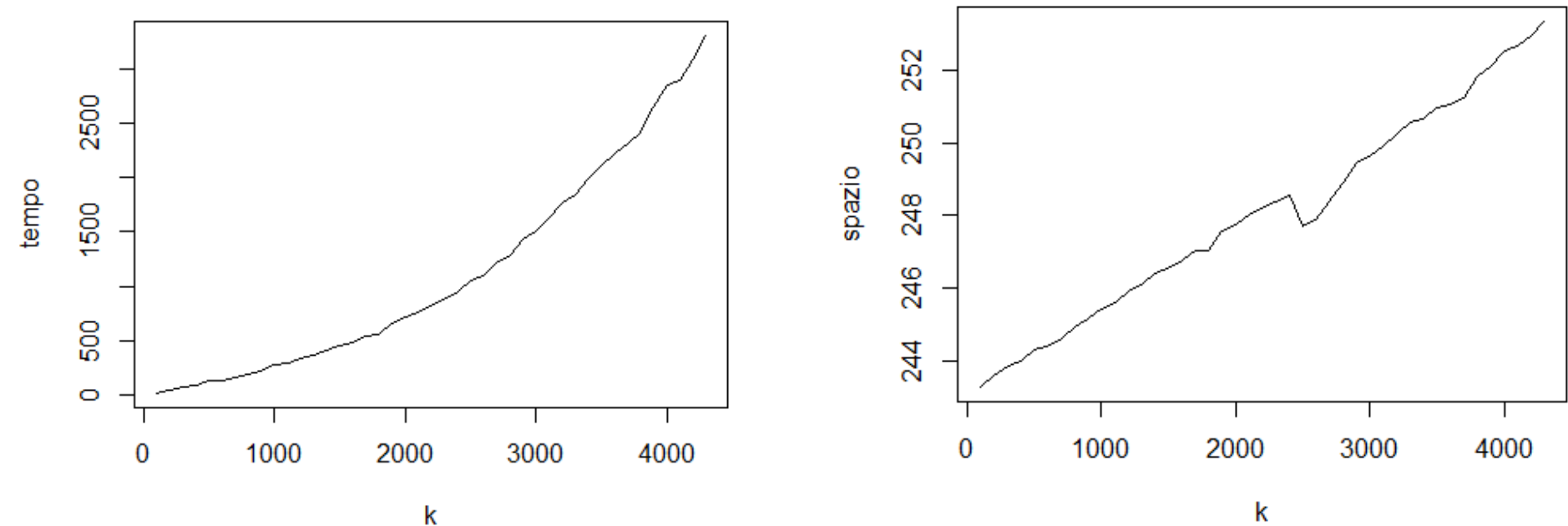


Figura 1: Tempo in secondi e spazio in GB della CUR al variare del parametro k tra 100 e 4300

Risultati: CUR accuratezza

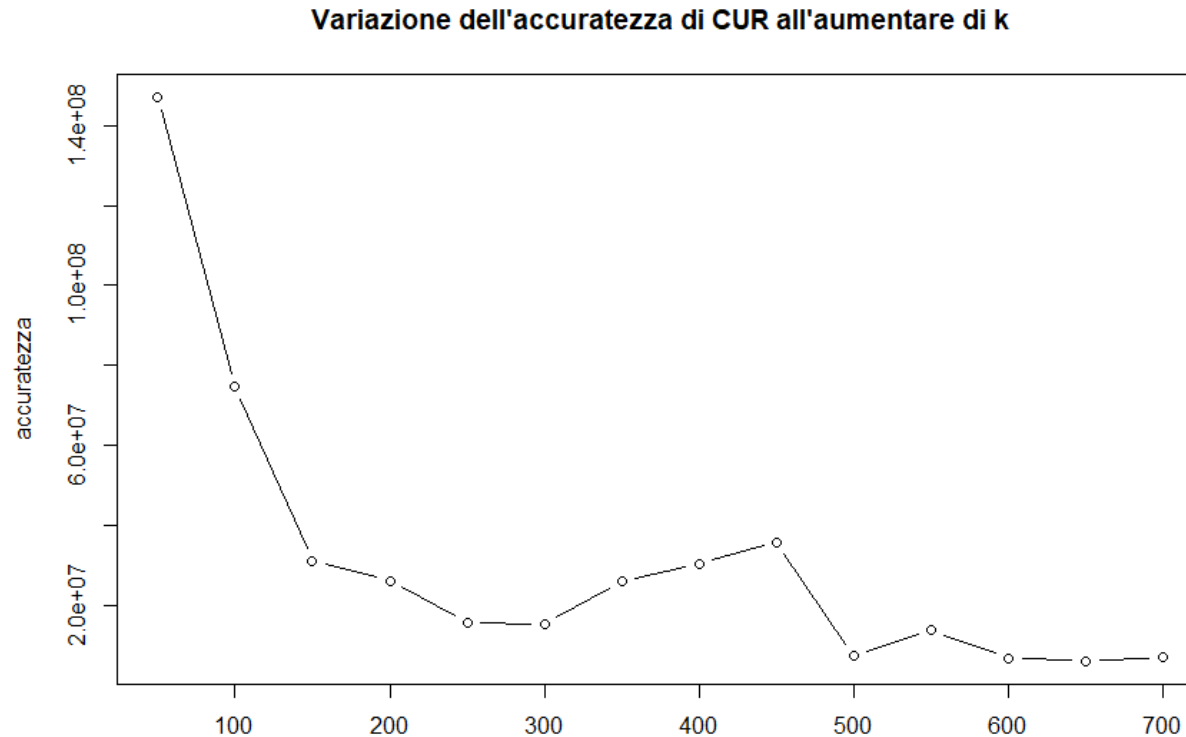


Figura 1: Accuratezza calcolata sulla approssimazione^k della CUR per la matrice A. Sono stati usati i dati di 9210 film

Risultati: spazio e tempo

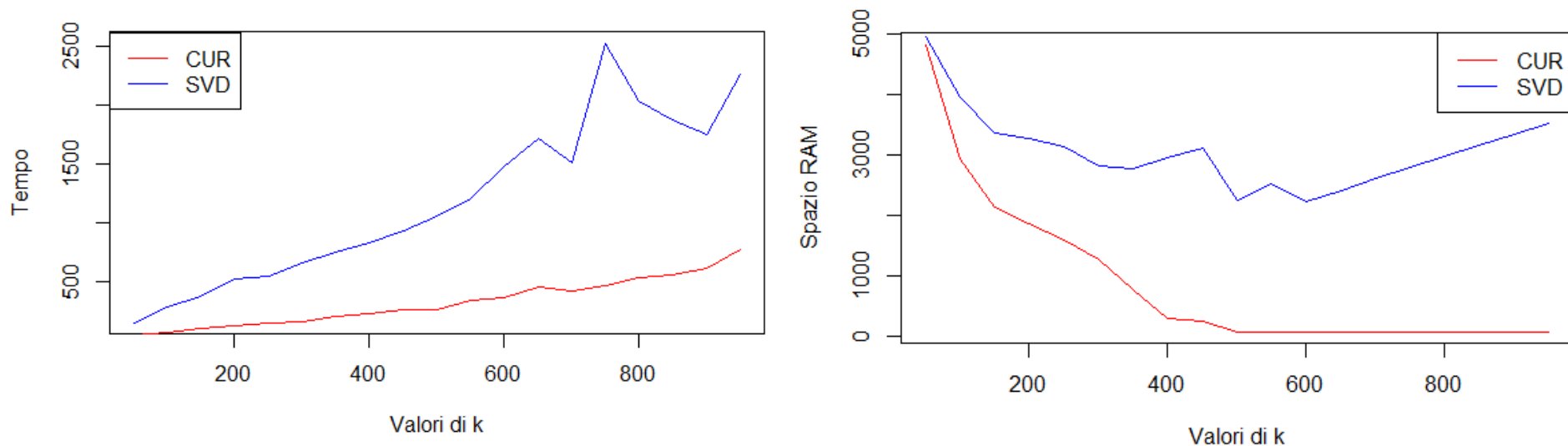


Figura 1: Variazioni del tempo e dello spazio di CUR e SVD al variare di k , mantenendo la dimensione dei dati costante e pari a 8000 film (circa 45.000.000 valutazioni)

Risultati: spazio e tempo

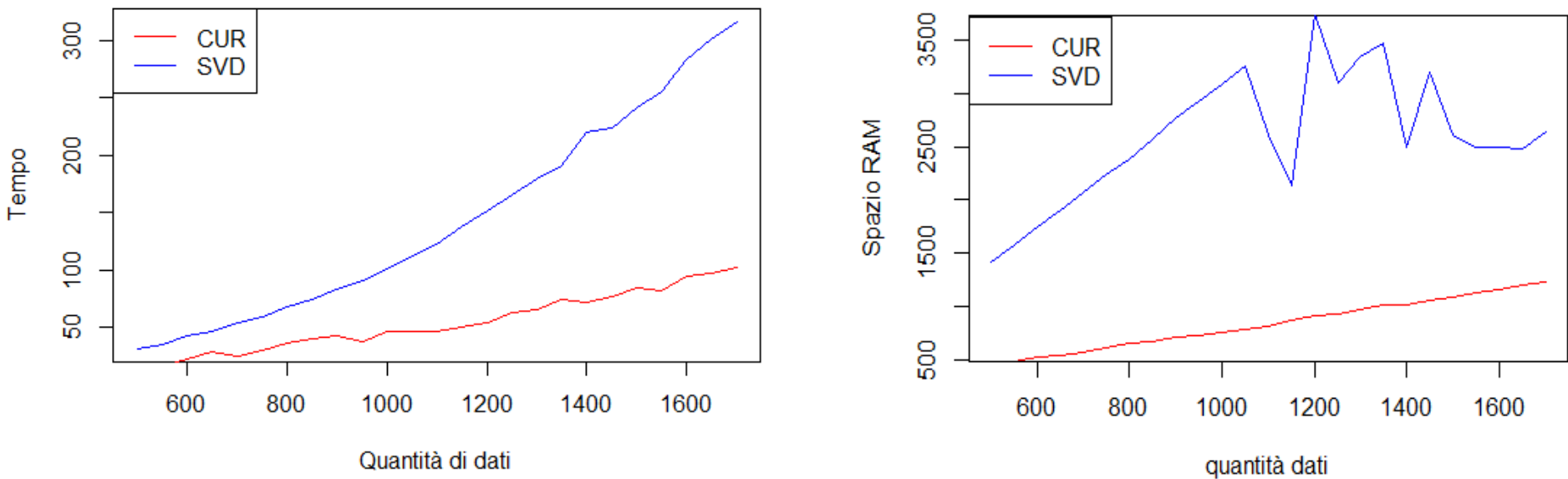


Figura 2: Variazioni del tempo e dello spazio di CUR e SVD al variare della quantità di dati mantenendo k costante, ogni 50 film aggiunti si hanno circa 267321 valutazioni in più

Risultati: accuratezza

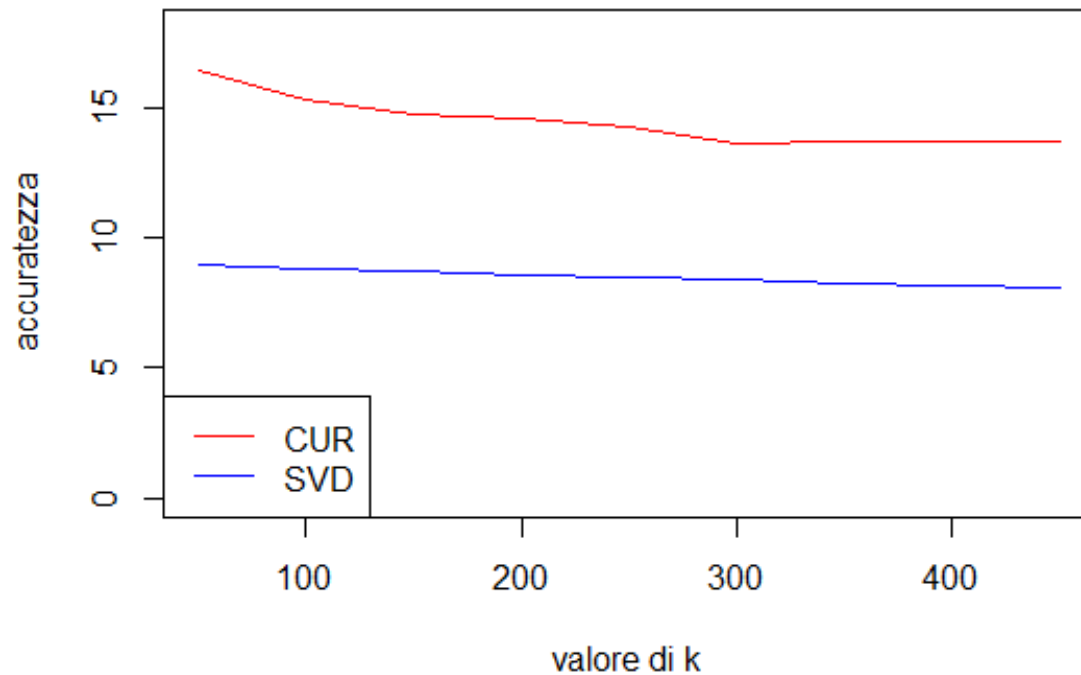


Figura 3: Accuratezza in scala logaritmica di CUR e SVD al variare di k mantenendo costante la quantità di dati pari a 2000

Risultati: accuratezza

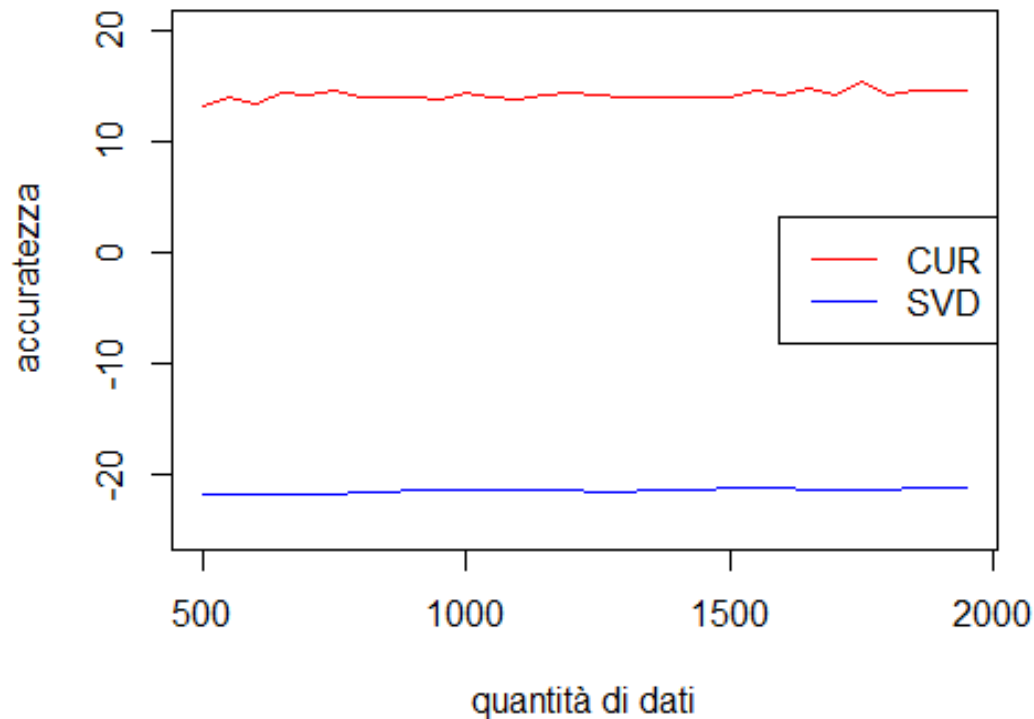


Figura 4: Accuratezza in scala logaritmica di CUR e SVD al variare di k mantenendo costante la quantità di dati pari a 2000

Conclusioni

❑ Cosa si è imparato?

- A livello pratico:
 - Gestione di dati di **grandi dimensioni**
 - Usare **linguaggi diversi** in contemporanea
- A livello teorico:
 - Imparato un **nuovo metodo** di decomposizione
 - Approfondito il metodo della SVD totale e troncata, approfondendo anche i **limiti di tali metodi**

❑ Difficoltà emerse?

- Il calcolo dell'accuratezza:
 - La **moltiplicazione di grandi matrici sparse**, di difficile implementazione. E' stato il problema che ha preso **più tempo in assoluto**
 - Calcolo della norma di Frobenius

Conclusioni

❑ Suggerimenti?

- Usare SQL è molto utile fino a quando si devono gestire dei dati, non lo è più quando si devono calcolare moltiplicazioni di matrici e differenze.
- Non aspettare le ultime fasi del codice per utilizzare tutto il dataset.

❑ Sviluppi futuri?

- Calcolare l'**accuratezza di CUR** con tutti e 4 i file, per **studiare al meglio** come questa migliori all'aumentare di r
- Provare effettivamente ad utilizzare nella pratica la decomposizione CUR, per effettuare delle raccomandazioni
- Provare a far girare il codice su **un computer più potente** in modo tale di avere in **meno tempo risultati migliori** e su una **quantità di dati maggiori**
- In generale cercare di **aumentare i dati utilizzati** per tutti gli esperimenti fatti

Grazie per l'attenzione

Miscellanea codice

Lettura del file

```
#Apri file di nome fname e inserisce i valori nel database di nome dbname
#Crea tabella valutazione(film,utente,val INT,PRIMARY KEY(film,utente)) e la riempie
def CreazioneValutazione(fname,dbname,cur):
    cur.execute("CREATE TABLE IF NOT EXISTS valutazione (film, utente, val INT, PRIMARY KEY(film,utente))")
    file=open(fname, mode='r')
    film=0
    utente=0
    val=0
    while lett:= file.readline():
        if ':' in lett :
            film= int(lett.replace(":\n",""))
        else:
            vect=lett.split(',')
            utente= int(vect[0])
            val=int(vect[1])
            cur.execute("insert into valutazione values (?, ?, ?)", (film, utente, val))
    print("Completata la creazione della tabella sql valutazione. Ultimo film inserito:", film)
    file.close()
```

Lettura del file

```
def CrValcontrol(fname, dbname, cur, MAXfilm, svdap=False):
    cur.execute("CREATE TABLE IF NOT EXISTS valutazione (film, utente, val INT, PRIMARY KEY(film,utente))")
    file=open(fname, mode='r')
    film=0
    utente=0
    val=0
    if svdap:
        filmA=[]
        valA=[]
        utenteA=[]
        while lett:= file.readline():
            if int(film)>int(MAXfilm):
                break
            if ':' in lett :
                film= int(lett.replace(":\n", ""))
            else:
                filmA.append(film)
                vect=lett.split(',')
                utente= int(vect[0])
                val=float(vect[1])
                valA.append(val)
                utenteA.append(utente)
                cur.execute("insert into valutazione values (?, ?, ?)", (film, utente, val))
        print("Completata la creazione della tabella sql valutazione. Ultimo film inserito:", film-1, ". Limite inserito:", MAXfilm, ".")
        file.close()
        return utenteA, filmA, valA

    while lett:= file.readline():
        if int(film)>int(MAXfilm):
            break
        if ':' in lett :
            film= int(lett.replace(":\n", ""))
        else:
            vect=lett.split(',')
            utente= int(vect[0])
            val=(vect[1])
            cur.execute("insert into valutazione values (?, ?, ?)", (film, utente, val))
    print("Completata la creazione della tabella sql valutazione. Ultimo film inserito:", film-1, ". Limite inserito:", MAXfilm, ".")
    file.close()
```

Creazione del database e indicizzazione dei dati

```
def FilmUtente(dbname, cur):
    cur.execute("SELECT SUM(val*val) FROM valutazione")
    SOMMATOT2= int(cur.fetchone()[0])
    cur.execute("CREATE TABLE IF NOT EXISTS utente(utente,somma2 INT,perc2 REAL, PRIMARY KEY(utente))")
    cur.execute('''INSERT INTO utente SELECT utente,SUM(val*val),cast(SUM(val*val) as REAL)/?
                  FROM valutazione GROUP BY utente''',(SOMMATOT2,))
    cur.execute("CREATE TABLE IF NOT EXISTS film(film,somma2 INT,perc2 REAL, PRIMARY KEY(film))")
    cur.execute('''INSERT INTO film SELECT film,SUM(val*val),cast(SUM(val*val) as REAL)/?
                  FROM valutazione GROUP BY film''',(SOMMATOT2,))
    print("Creato film e utente!")
```

Dizionari

```
def dizionari(curs,svdap=False):
    #iniziate la tabelle su sql e i dizionari
    curs.execute("CREATE TABLE IF NOT EXISTS indiceU(utente,indice, PRIMARY KEY(utente))")
    curs.execute("CREATE TABLE IF NOT EXISTS indiceF(film,indice, PRIMARY KEY(film))")
    dizionarioUte={}
    dizionarioFilm={}
    if svdap:
        dizionarioUteInd={}

    #si riempiono il dizionario e la tabella relative ad utente
    curs.execute("SELECT utente, perc2 FROM utente")
    ute=curs.fetchall()
    indiceU=0
    if svdap:
        for u in ute:
            curs.execute("INSERT INTO indiceU values(?,?)", (u[0],indiceU))
            dizionarioUteInd[u[0]]=indiceU
            dizionarioUte[u[0]]=float(u[1])
            indiceU=indiceU+1
    else:
        for u in ute:
            curs.execute("INSERT INTO indiceU values(?,?)", (u[0],indiceU))
            dizionarioUte[u[0]]=float(u[1])
            indiceU=indiceU+1

    #si riempiono il dizionario e la tabella relative a film
    curs.execute("SELECT film, perc2 FROM film")
    indiceF=0
    fil=curs.fetchall()
    for f in fil:
        curs.execute("INSERT INTO indiceF values(?,?)", (f[0],indiceF))
        dizionarioFilm[f[0]]=float(f[1])
        indiceF=indiceF+1

    print("Effettuata la creazione dei dizionari.")
    if svdap:
        print("Effettuata anche la creazione del dizionario per l'indice degli utenti.")
        return dizionarioUte,dizionarioFilm,dizionarioUteInd
    return dizionarioUte,dizionarioFilm
```

1. La scelta

```
#A dal numero k di elementi da scegliere e dai dizionari,
#ritorna k utenti e k film che vengono scelti casualmente
#seguendo una distribuzione multinomiale
def sceltak(k,dizionarioUte,dizionarioFilm):
    #A partire dai dizionari crea 4 np.array, necessari per la scelta casuale
    ArUte=np.array(list(dizionarioUte.keys()))
    ArPer=np.array(list(dizionarioUte.values()))
    AcFil=np.array(list(dizionarioFilm.keys()))
    AcPer=np.array(list(dizionarioFilm.values()))

    #vengono scelti k utenti e k film seguendo una distribuzione multinomiale.
    #Le probabilità sono dati dagli oggetti dei dizionari
    kutenti=np.random.choice(ArUte, size=k, replace=True, p=ArPer)
    kfilm=np.random.choice(AcFil, size=k, replace=True, p=AcPer)

    print("Effettuata la scelta dei",k,"elementi.")
    return kutenti,kfilm
```


2. Calcolo di C

```
#Dando il numero k di elementi necessari, il dizionario riferito a film e il vettore dei k film scelti,
#viene calcolata la matrice C in SQL, se svd è True viene ritornata C anche in matrice.
def calcC(k,curs,con,dizionarioFilm,kfilm,svd=False):
    curs.execute("CREATE TABLE IF NOT EXISTS c(utenteI INT,filmI INT,val REAL)")
    curs.execute("SELECT COUNT(DISTINCT utente) FROM valutazione")
    mm=curs.fetchone()[0]
    #per ogni film scelto viene prese la colonna corrispondente e
    #il valore di valutazione viene diviso per sqrt(dizionarioFilm.get(kfilm[i])*k)
    for i in range(k):
        curs.execute('''INSERT INTO c SELECT indice, ?, CAST(val AS REAL)/? FROM indiceU
                        NATURAL JOIN (SELECT val, utente FROM valutazione WHERE film=?)''',
                        (i,math.sqrt(dizionarioFilm.get(kfilm[i])*k),int(kfilm[i])))
    curs.execute("CREATE INDEX cI ON c (utenteI,filmI)")
    con.commit()
    print("Calcolata C, è contenuta nella tabella sql c.")
    if svd:
        indUteC=[]
        indFilmC=[]
        valC=[]

        curs.execute("SELECT * FROM c")
        cc=curs.fetchall()
        for c in cc:
            indUteC.append(c[0])
            indFilmC.append(c[1])
            valC.append(float(c[2]))
        C=coo_matrix((valC,(indUteC,indFilmC)),shape=(mm,k))
    return C
```

2. Calcolo di R

```
#Dando il numero k di elementi necessari, il dizionario riferito a utenti e
#il vettore dei k utenti scelti, viene calcolata la matrice R in SQL,
#se svd è True viene ritornata C anche in matrice.
def calcR(k,curs,con,dizionarioUte,kutenti,svd=False):
    curs.execute("CREATE TABLE IF NOT EXISTS r(utenteI INT,filmI INT,val REAL)")
    #per ogni utente scelto viene presa la colonna corrispondente e il valore di valutazione
    #viene diviso per sqrt(dizionarioUte.get(kutenti[i])*k)
    for i in range(k):
        curs.execute('''INSERT INTO r SELECT ?, indice, CAST(val AS REAL)/? FROM indiceF
                        NATURAL JOIN (SELECT val, film FROM valutazione WHERE utente=? )''',
                      (i,math.sqrt(dizionarioUte.get(kutenti[i])*k),int(kutenti[i])))
    curs.execute("CREATE INDEX rI ON r (utenteI,filmI)")
    print("Calcolata R, è contenuta nella tabella sql r.")
    if svd:
        indUteR=[]
        indFilmR=[]
        valR=[]

        curs.execute("SELECT * FROM r")
        rr=curs.fetchall()
        for r in rr:
            indUteR.append(r[0])
            indFilmR.append(r[1])
            valR.append(float(r[2]))
        curs.execute("SELECT COUNT(DISTINCT film) FROM valutazione")
        mm=curs.fetchone()[0]
        R=coo_matrix((valR,(indUteR,indFilmR)),shape=(k,mm))
        return R
    con.commit()
```

2. Calcolo di U

```
#Dando il numero k, gli utenti e i film scelti viene calcolata la matrice U e viene inserita in SQL nella tabella u.
#se svd=True viene riportata la matrice U
def calcU(k,curs,con,kutenti,kfilm,svd=False):
    #Si costruisce la matrice W come matrice coo_matrix.
    indUteW=[]
    indFilmW=[]
    valW=[]

    for i in range(k):
        for j in range(k):
            curs.execute("SELECT val FROM valutazione WHERE utente=? AND film=?", (int(kutenti[j]),int(kfilm[i])))
            cc=curs.fetchone()
            if cc!=None:
                indUteW.append(j)
                indFilmW.append(i)
                valW.append(float(cc[0]))
    W=coo_matrix((valW,(indUteW,indFilmW)),shape=(k,k))

    #Si effettua la decomposizione SVD sulla matrice W
    X,sigma,Yt=svds(W, k=k-1, ncv=None, tol=0, which='LM', v0=None, maxiter=None,
                    return_singular_vectors=True, solver='arpack', random_state=None, options=None)

    #Si calcola la psuedoinversa
    psSigma=np.linalg.pinv(np.diag(sigma))
    #Si calcola la matrice U
    del W
    U=coo_matrix(X @ psSigma @ Yt)
    del X,psSigma,Yt
    if svd:
        print("Calcolata U")
        return U
    #Caricata su SQL nella tabella u
    curs.execute("CREATE TABLE IF NOT EXISTS u(utenteI INT,filmI INT,val REAL, PRIMARY KEY(utenteI,filmI))")
    for i,j,v in zip(U.row,U.col,U.data):
        curs.execute("INSERT INTO u values(?,?,?)", (int(i),int(j),v))
    print("Calcolata U")
    con.commit()
```



Obiettivo 1: Calcolo della CUR

```
def dividiA(curs):
    #Trova i valori mediani del numero di film e utenti
    curs.execute("SELECT max(indice) FROM indiceU")
    uMAX=int(curs.fetchone()[0])+1
    curs.execute("SELECT max(indice) FROM indiceF")
    fMAX=int(curs.fetchone()[0])+1
    #Troviamo A1
    indUteA1=[]
    indFilmA1=[]
    valA1=[]
    curs.execute('''SELECT indiceU.indice,indiceF.indice,valutazione.val FROM valutazione JOIN indiceU
                    ON valutazione.utente=indiceU.utente JOIN indiceF ON valutazione.film=indiceF.film
                    WHERE indiceU.indice<? and indiceF.indice<?''',(int(uMAX/2),int(fMAX/2)))
    cc=curs.fetchall()
    for c in cc:
        indUteA1.append(c[0])
        indFilmA1.append(c[1])
        valA1.append(float(c[2]))
    A1=coo_matrix((valA1,(indUteA1,indFilmA1)),shape=(int(uMAX/2),int(fMAX/2)))
    del valA1,indUteA1,indFilmA1
    #Troviamo A2
    indUteA2=[]
    indFilmA2=[]
    valA2=[]
    curs.execute('''SELECT indiceU.indice,indiceF.indice,valutazione.val FROM valutazione JOIN indiceU
                    ON valutazione.utente=indiceU.utente JOIN indiceF ON valutazione.film=indiceF.film
                    WHERE indiceU.indice<? and indiceF.indice>?''',(int(uMAX/2),int(fMAX/2)-1))
    cc=curs.fetchall()
    for c in cc:
        indUteA2.append(c[0])
        indFilmA2.append(c[1]-int(fMAX/2))
        valA2.append(float(c[2]))
    A2=coo_matrix((valA2,(indUteA2,indFilmA2)),shape=(int(uMAX/2),fMAX-int(fMAX/2)))
    del valA2,indUteA2,indFilmA2
```



Obiettivo 1: Calcolo della CUR

```
def trovaC1C2(curs, k, uMAX) :  
    indUteC1=[]  
    indFilmC1=[]  
    valC1=[]  
    indUteC2=[]  
    indFilmC2=[]  
    valC2=[]  
    #Trova C1  
    curs.execute("SELECT * FROM c WHERE utenteI<? and filmI<?", (int(uMAX/2),int(k/2)))  
    cc=curs.fetchall()  
    for c in cc:  
        indUteC1.append(c[0])  
        indFilmC1.append(c[1])  
        valC1.append(float(c[2]))  
    C1=coo_matrix((valC1, (indUteC1, indFilmC1)), shape=(int(uMAX/2), int(k/2)))  
    del valC1, indUteC1, indFilmC1  
    #Trova C2  
    curs.execute("SELECT * FROM c WHERE utenteI<? and filmI>?", (int(uMAX/2),int(k/2)-1))  
    cc=curs.fetchall()  
    for c in cc:  
        indUteC2.append(c[0])  
        indFilmC2.append(c[1]-int(k/2))  
        valC2.append(float(c[2]))  
    C2=coo_matrix((valC2, (indUteC2, indFilmC2)), shape=(int(uMAX/2), k-int(k/2)))  
    del valC2, indUteC2, indFilmC2  
    return C1, C2
```

Obiettivo 2: Lettura del file

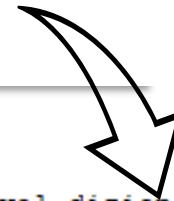
```
*ApVal.py - C:\Users\elena\Desktop\ApVal.py (3.10.9)*
File Edit Format Run Options Window Help

#Funzione che crea la tabella valutazione fermandosi al film di numero MAXfilm
#se svdap=True allora ritorna anche i vettori filmA,utenteA e valA necessari per la SVD
def CrValcontrol(fname,dbname,cur,MAXfilm,svdap=False):
    cur.execute("CREATE TABLE IF NOT EXISTS valutazione (film, utente, val INT, PRIMARY KEY(film,utente))")
    file=open(fname, mode='r')
    film=0
    utente=0
    val=0
    if svdap:
        filmA=[]
        valA=[]
        utenteA=[]
        while lett:= file.readline():
            if int(film)>int(MAXfilm):
                break
            if ':' in lett :
                film= int(lett.replace(":\n",""))
            else:
                filmA.append(film)
                vect=lett.split(',')
                utente= int(vect[0])
                val=float(vect[1])
                valA.append(val)
                utenteA.append(utente)
                cur.execute("insert into valutazione values (?, ?, ?)", (film, utente, val))
        print("Completata la creazione della tabella sql valutazione. Ultimo film inserito:", film-1, ". Limite inserito:",MAXfilm, ".")
        file.close()
        return utenteA, filmA, valA

    while lett:= file.readline():
        if int(film)>int(MAXfilm):
            break
        if ':' in lett :
            film= int(lett.replace(":\n",""))
        else:
            vect=lett.split(',')
            utente= int(vect[0])
            val=(vect[1])
            cur.execute("insert into valutazione values (?, ?, ?)", (film, utente, val))
    print("Completata la creazione della tabella sql valutazione. Ultimo film inserito:", film-1, ". Limite inserito:",MAXfilm, ".")
    file.close()
```

Obiettivo 2: Costruzione di A

```
if svdap:
    for u in ute:
        curs.execute("INSERT INTO indiceU values(?,?)", (u[0], indiceU))
        dizionarioUteInd[u[0]]=indiceU
        dizionarioUte[u[0]]=float(u[1])
        indiceU=indiceU+1
else:
```



```
def costruisciA(utente, film, val, dizionarioUteInd):
    indUte=[]
    vaL=np.array(val)
    indFilm=np.array(film)
    print(len(vaL), len(indFilm))
    for u in utente:
        indUte.append(dizionarioUteInd.get(u))
    #Costruisce la matrice A come matrice sparsa
    AC=coo_matrix((vaL, (indUte, indFilm)))
    return AC
```



Obiettivo 2:

SVD totale VS CUR

```
def calcolaSVD(cur, AC, SVDvec=False):  
    t = Timer(name="class")  
    A=AC.todense()  
    print("fatta A.")  
    t.start()  
    #Calcola la SVD  
    X, sigma, Yt=np.linalg.svd(A, full_matrices=False)  
    fine=t.stop()  
    SVD=coo_matrix(X @ np.diag(sigma) @ Yt)  
    spa=psutil.Process(os.getpid()).memory_info().rss//(1024.0**2)  
    del X, sigma, Yt  
    diff=norm(AC-SVD, 'fro')  
    if SVDvec:  
        return fine, X, sigma, Yt, diff, spa  
    return fine, diff, spa
```

```
def calcolaCURdist(A, k, curs, con, dizionarioUte, dizionarioFilm):  
    t = Timer(name="class")  
    t.start()  
    kutenti, kfilm=sceltak(k, dizionarioUte, dizionarioFilm)  
  
    C=calcC(k, curs, con, dizionarioFilm, kfilm, True)  
    R=calcR(k, curs, con, dizionarioUte, kutenti, True)  
    U=calcU(k, curs, con, kutenti, kfilm, True)  
    fine=t.stop()  
  
    con.commit()  
  
    CUR=coo_matrix(C.dot(U).dot(R))  
    del C, U, R  
    distanza=norm(CUR-A, 'fro')  
  
    return fine, distanza
```




Obiettivo2:

SVD totale VS CUR

```
def calcolaSVD(cur, AC, SVDvec=False):  
    t = Timer(name="class")  
    A=AC.todense()  
    print("fatta A.")  
    t.start()  
    #Calcola la SVD  
    X,sigma,Yt=np.linalg.svd(A,full_matrices=False)  
    fine=t.stop()  
    spa=psutil.Process(os.getpid()).memory_info().rss/(1024**2)  
    if SVDvec:  
        return fine,X,sigma,Yt,diff  
    return fine, spa
```

```
def calcolaCUR(k, curs, con, dizionarioUte, dizionarioFilm):  
  
    t = Timer(name="class")  
    t.start()  
    kutenti,kfilm=sceltak(k,dizionarioUte,dizionarioFilm)  
    calcC(k,curs,con,dizionarioFilm,kfilm)  
    calcR(k,curs,con,dizionarioUte,kutenti)  
    calcU(k,curs,con,kutenti,kfilm)  
    fine=t.stop()  
    spa=psutil.Process(os.getpid()).memory_info().rss/(1024**2)  
  
    return fine,spa
```



Obiettivo 2:

SVD parziale VS CUR

```
def calcolaSVDk(cur,A,k):  
    t = Timer(name="class")  
    t.start()  
    #Calcola la SVD  
    X,sigma,Yt=svds(A, k=k, ncvt=None, tol=0, which='LM',  
                    v0=None, maxiter=None, return_singular_vectors=True,  
                    solver='arpack', random_state=None, options=None)  
  
    fine=t.stop()  
    SVD=coo_matrix(X.dot(np.diag(sigma)).dot(Yt))  
    diff=norm(A-SVD,'fro')  
  
    return fine,diff
```

```
def calcolaCURdist(A,k,curs,con,dizionarioUte,dizionarioFilm):  
    t = Timer(name="class")  
    t.start()  
    kutenti,kfilm=sceltak(k,dizionarioUte,dizionarioFilm)  
  
    C=calcC(k,curs,con,dizionarioFilm,kfilm,True)  
    R=calcR(k,curs,con,dizionarioUte,kutenti,True)  
    U=calcU(k,curs,con,kutenti,kfilm,True)  
    fine=t.stop()  
  
    con.commit()  
  
    CUR=coo_matrix(C.dot(U).dot(R))  
    del C,U,R  
    distanza=norm(CUR-A, 'fro')  
  
    return fine,distanza
```

- ❑ Per calcolare l'accuratezza viene usata la funzione `scipy.sparse.linalg.norm()`, che calcola la distanza di Frobenius in questo modo:

$$\sqrt{\sum_{i,j} (a_{ij} - x_{ij})^2}$$



Obiettivo2: SVD parziale VS CUR

```
def calcolaSVDk(cur,A,k):  
    t = Timer(name="class")  
    t.start()  
    #Calcola la SVD  
    X,sigma,Yt=svds(A, k=k, ncv=None, tol=0, which='LM',  
                    v0=None, maxiter=None, return_singular_vectors=True,  
                    solver='arpack', random_state=None, options=None)  
    fine=t.stop()  
    spa=psutil.Process(os.getpid()).memory_info().rss/(1024**2)  
  
    return fine,spa
```

```
def calcolaCUR(k,curs,con,dizionarioUte,dizionarioFilm):  
  
    t = Timer(name="class")  
    t.start()  
    kutenti,kfilm=sceltak(k,dizionarioUte,dizionarioFilm)  
    calcC(k,curs,con,dizionarioFilm,kfilm)  
    calcR(k,curs,con,dizionarioUte,kutenti)  
    calcU(k,curs,con,kutenti,kfilm)  
    fine=t.stop()  
    spa=psutil.Process(os.getpid()).memory_info().rss/(1024**2)  
  
    return fine,spa
```



Obiettivo 1: Calcolo della CUR

```
if __name__ == '__main__':
#Creazione Database
    dbname= 'Zdatabasedef.db'
    con = sqlite3.connect(dbname)
    curs = con.cursor()
    ApVal.CreazioneValutazione("combined_data_1.txt",dbname,curs)
    ApVal.CreazioneValutazione("combined_data_2.txt",dbname,curs)
    ApVal.FilmUtente(dbname,curs)
    con.commit()

#Creazione dei dizionari
    dizionarioUte,dizionarioFilm=curr.dizionari(curs)

#Apertura file, suddivido A
    fx = open("AccTempoTOT.txt","w")
    t = Timer(name="class")
    tempok=0
    frobk=0
    A1,A2,A3,A4,uMAX,fMAX=curr.dividiA(curs)

#Al variare di r calcolo la CUR e l'accuratezza
    for k in range(50,10000000,50):
        t.start()
        tempok=curr.calcolaCUR(k,curs,con,dizionarioUte,dizionarioFilm)
        frobk=curr.prodottoCURpy(curs,A1,A2,A3,A4,k,uMAX,fMAX)
        print(k,tempok,frobk,psutil.disk_usage('/').used/(2**30),file=fx)
        fx.flush()
        curs.execute("DROP TABLE IF EXISTS cur")
        curr.dropCUR(curs)
        con.commit()
        curs.execute("VACUUM")
        print("Per fare",k,"vettori c'è voluto",t.stop())
    print("Fatto CUR")
    fx.close()
    con.commit()
    con.close()
```

r	tempos	acc	spDiscoGb
50	35.855490	38759758.505	244.20456
100	61.723308	21931331.094	244.26503

Utilizziamo i dati relativi a 9210
film (~51.040.567 entrate).
A è una matrice da
480.189x9210.



Obiettivo 1: Calcolo della CUR

```
if __name__ == '__main__':
#Creazione Database
    dbname= 'Zdatabasedef.db'
    con = sqlite3.connect(dbname)
    curs = con.cursor()
    ApVal.CreazioneValutazione("combined_data_1.txt", dbname, curs)
    ApVal.CreazioneValutazione("combined_data_2.txt", dbname, curs)
    ApVal.CreazioneValutazione("combined_data_3.txt", dbname, curs)
    ApVal.CreazioneValutazione("combined_data_4.txt", dbname, curs)
    ApVal.FilmUtente(dbname, curs)
    con.commit()

#Creazione dei dizionari
    dizionarioUte, dizionarioFilm=curr.dizionari(curs)

#Apertura file, suddivido A
    fx = open("AccTempoTOT.txt", "w")
    t = Timer(name="class")
    tempok=0

#Al variare di r calcolo la CUR e l'accuratezza
    for k in range(50,10000000,50):
        t.start()
        tempok=curr.calcolaCUR(k, curs, con, dizionarioUte, dizionarioFilm)
        print(k, tempok, psutil.disk_usage('/').used/(2**30), file=fx)
        fx.flush()
        curr.dropCUR(curs)
        con.commit()
        curs.execute("VACUUM")
        print("Per fare", k, "vettori c'è voluto", t.stop())
    print("Fatto CUR")
    fx.close()
    con.commit()
    con.close()
```

r	tempos	SpRAMGb
400	96.08881490	244.0037078857422
500	130.3511207	244.31463623046875

Utilizziamo i dati relativi
a tutti i film
(~100.480.511 entrate).
A è una matrice da
480.189x17.770.



Obiettivo 2: SVD totale VS CUR

```
if __name__ == '__main__':
    #Creazione Database
    dbname= 'FdatabaseSVD.db'
    con = sqlite3.connect(dbname)
    curs = con.cursor()

    #Apertura file
    fx = open("testSVD.txt", "w")
    tempokc=0
    frobkc=0
    tempoks=0
    frobks=0
    for t in range(500,2000,50):
        #Costruzione Database
        utenteA, filmA, valA=ApVal.CrValcontrol("combined_data_1.txt", dbname, curs, t, True)
        ApVal.FilmUtente(dbname, curs)
        con.commit()
        dizionarioUte, dizionarioFilm, dizionarioUteInd=curr.dizionari(curs, True)
        A=ApVal.costruisciA(utenteA, filmA, valA, dizionarioUteInd)
        #Calcolo CUR
        tempokc, frobkc, spac=curr.calcolaCURdist(A, int(t/10), curs, con, dizionarioUte, dizionarioFilm)
        print("CUR", t, tempokc, frobkc, spac, file=fx)
        fx.flush()
        curs.execute("DROP TABLE IF EXISTS cur")
        curr.dropCUR(curs)
        con.commit()
        curs.execute("VACUUM")
        #Calcolo SVD
        tempoks, frobks, spas=svdii.calcolaSVD(curs, A)
        print("SVD", t, tempoks, frobks, spas, file=fx)
        #Svuoto il dataframe
        del A
        curs.execute("DROP TABLE IF EXISTS valutazione")
        curs.execute("DROP TABLE IF EXISTS indiceU")
        curs.execute("DROP TABLE IF EXISTS indiceF")
        curs.execute("DROP TABLE IF EXISTS utente")
        curs.execute("DROP TABLE IF EXISTS film")
        con.commit()
        curs.execute("VACUUM")
        print("Effettuato il test con", t, "film.")
    fx.close()
    con.commit()
    con.close()
```

Cos	nfi	tempokc	accur	SpRAMMb
CUR	800	21.4833870999	1293539.26080	2.0
SVD	800	52.9158998	3.62430326e-10	2406.0

Per ogni 50 film vengono
aggiunte ~ **267.321** entrate.
Rendendo A una matrice
sempre più grande.



Obiettivo2:

SVD totale VS CUR

```
if __name__ == '__main__':
#Creazione Database
    dbname= 'FdatabaseSVD.db'
    con = sqlite3.connect(dbname)
    curs = con.cursor()
#test di ottimizzazione CUR efficienza e efficacia
    fx = open("iltestSVD.txt", "w")
    for t in range(500,5000,50):
        #Creiamo il database con t film
        utenteA, filmA, valA=ApVal.CrValcontrol("C:\\Users\\Utente\\Desktop\\combined_data_1.txt",
                                                dbname, curs, t, True)

        ApVal.FilmUtente(dbname, curs)
        con.commit()
        dizionarioUte, dizionarioFilm, dizionarioUteInd=curr.dizionari(curs, True)
        A=ApVal.costruisciA(utenteA, filmA, valA, dizionarioUteInd)
        #Calcoliamo la CUR
        tempokc, spac=curr.calcolaCUR(int(t/10), curs, con, dizionarioUte, dizionarioFilm)
        print("CUR", t, tempokc, spac, file=fx)
        fx.flush()
        curs.execute("DROP TABLE IF EXISTS cur")
        curr.dropCUR(curs)
        con.commit()
        curs.execute("VACUUM")
        #calcoliamo la SVD
        tempoks, spas=svdii.calcolaSVD(curs, A)
        print("SVD", t, tempoks, spas, file=fx)
        #Togliamo tutto
        curs.execute("DROP TABLE IF EXISTS valutazione")
        curs.execute("DROP TABLE IF EXISTS indiceU")
        curs.execute("DROP TABLE IF EXISTS indiceF")
        curs.execute("DROP TABLE IF EXISTS utente")
        curs.execute("DROP TABLE IF EXISTS film")
        con.commit()
        curs.execute("VACUUM")
        print("Effettuato il test con", t, "film.")
    fx.close()
    con.commit()
    con.close()
```

Cos	nfi	tempos	SpRAMMb
CUR	600	42.4352330998518	523.375
SVD	600	66.7193646999076	1747.644

Lo lasciamo eseguire fino a quando la SVD non si blocca, viene riportato il codice **init_gesdd failed init.**

Si blocca a 1700 film, quando A è una matrice 440.495x1.700 e ci sono 8.502.250 entrate.





Obiettivo 2: SVD parziale VS CUR

```
if __name__ == '__main__':
```

```
#Creazione Database
```

```
dbname= 'databasedefSVDk.db'  
con = sqlite3.connect(dbname)  
curs = con.cursor()
```

```
fx = open("test2SVD.txt","w")  
tempokc=0  
frobkc=0  
tempoks=0  
frobks=0
```

```
# Riempio il database
```

```
utenteA, filmA, valA=ApVal.CrValcontrol("combined_data_1.txt", dbname, curs, 2000, True)  
ApVal.FilmUtente(dbname, curs)  
con.commit()
```

```
dizionarioUte, dizionarioFilm, dizionarioUteInd=curr.dizionari(curs, True)
```

```
# Calcolo A
```

```
AC=ApVal.costruisciA(utenteA, filmA, valA, dizionarioUteInd)
```

```
for k in range(50, 500, 50):
```

```
# Calcolo la CUR
```

```
tempokc, frobkc=curr.calcolaCURdist(AC, k, curs, con, dizionarioUte, dizionarioFilm)  
print("CUR", k, tempokc, frobkc, file=fx)  
fx.flush()  
curr.dropCUR(curs)  
con.commit()  
curs.execute("VACUUM")
```

```
# Calcolo la SVD parziale
```

```
tempoks, frobks=svdii.calcolaSVDk(curs, AC, k)  
print("SVD", k, tempoks, frobks, file=fx)  
fx.flush()  
con.commit()  
curs.execute("VACUUM")  
print("Effettuato il test per", k, ".")
```

```
fx.close()  
con.commit()  
con.close()
```

	Cos	k	tempos	acc
CUR	50	37.79248890001327	13506179.231393535	
SVD	50	14.28081110003404	7895.647753602244	

In questo caso utilizziamo solo i dati relativi a **2.000 film** (~10.317.269 entrate).
A è una matrice da **449.098x2000**.



Obiettivo2:

SVD parziale VS CUR

```
if __name__ == '__main__':
```

```
#Creazione Database
```

```
dbname= 'CdatabasedefSVDk.db'  
con = sqlite3.connect(dbname)  
curs = con.cursor()
```

```
fx = open("test2SVD.txt", "w")  
tempokc=0  
frobkc=0  
tempoks=0  
frobks=0
```

```
# Riempio il database
```

```
utenteA, filmA, valA=ApVal.CrValcontrol("C:\\Users\\Utente\\Desktop\\combined_data_1.txt", dbname,  
                                         curs, 5000, True)  
utenteA, filmA, valA=ApVal.CrValcontrol2("C:\\Users\\Utente\\Desktop\\combined_data_2.txt", dbname,  
                                          curs, 8000, filmA, valA, utenteA)  
  
ApVal.FilmUtente(dbname, curs)  
con.commit()
```

```
dizionarioUte, dizionarioFilm, dizionarioUteInd=curr.dizionari(curs, True)
```

```
# Calcolo A
```

```
AC=ApVal.costruisciA(utenteA, filmA, valA, dizionarioUteInd)
```

```
for k in range(50, 1000, 50):
```

```
# Calcolo la CUR
```

```
tempokc, spac=curr.calcolaCURdist(AC, k, curs, con, dizionarioUte, dizionarioFilm)  
print("CUR", k, tempokc, spac, file=fx)  
fx.flush()  
curr.dropCUR(curs)  
con.commit()  
curs.execute("VACUUM")
```

```
# Calcolo la SVD parziale
```

```
tempoks, spas=svdii.calcolaSVDk(curs, AC, k)  
print("SVD", k, tempoks, spas, file=fx)  
fx.flush()  
con.commit()  
curs.execute("VACUUM")  
print("Effettuato il test per", k, ".")
```

```
fx.close()  
con.commit()
```

	Cos	k	tempos	SpRAMMb
CUR	250	560.7524213001598	465.3828125	
SVD	250	392.0643120999448	1739.7890625	

In questo caso utilizziamo solo i dati relativi a **8.000 film** (~45.000.000 entrate).
A è una matrice da **463.314x8000**.

