



Università Degli Studi di Perugia

Giacomo Calabresi - Mat. 318286

Raccomandazione / Filtering collaborativo basato su Random Walk

Interactive Intelligent Devices, Systems And Environments

Dipartimento di Matematica e Informatica
Intelligent and Mobile Computing



2019

© Giacomo Calabresi - Mat. 318286, 2019

*Series of dissertations submitted to the
Intelligent and Mobile Computing, Università Degli Studi di Perugia*

All rights reserved. No part of this publication may be
reproduced or transmitted, in any form or by any means, without permission.

Indice

Indice	i
1 Introduzione	1
2 Filtering Collaborativo e Random Walk	3
3 Soluzione proposta	7
4 Conclusioni	11
A Appendice	13

Capitolo 1

Introduzione

Il presente testo illustra una delle possibili metodologie per implementare un sistema di raccomandazioni (*recommender system*).

La tecnica scelta è conosciuta come **Random Walk** ed è stata applicata a un dataset di pellicole cinematografiche¹.

Lo scopo quindi è semplice: ricavare consigli su nuovi film da vedere in base ai voti (*ratings*) assegnati dagli altri utenti, che hanno già potuto visionarli.

Ciò avverrà estrapolando un cammino (pseudo)casuale da un certo grafo dove i nodi sono gli utenti e i film e un arco è presente se un certo utente ha visto (e valutato) un certo film.

🔴 **Giacomo Calabresi - Mat. 318286**

Perugia, October 2019

¹<https://grouplens.org/datasets/movielens/>

Capitolo 2

Filtering Collaborativo e Random Walk

L'idea è quindi di adottare un approccio al problema tramite **filtering collaborativo**, il quale permette di generare in modo automatico delle predizioni per un certo utente, basandosi su dati di altri utenti che hanno espresso gradimento per determinati oggetti.

Restando nel campo dei film: poniamo di avere 2 utenti, A e B, se ai film che hanno visto entrambi hanno assegnato voti simili, possiamo ipotizzare che, per un dato film visto da A ma non da B, quest'ultimo (quando lo visionerà) possa assegnare un voto simile a quello lasciato da A. In caso di voto alto da parte di A, B potrebbe quindi in linea di massima essere interessato, pertanto un *recommender system* potrebbe suggerire quel film. Naturalmente, maggiore è la dimensione del dataset (quindi più associazioni $[nome_film - voto]$ sono presenti da parte di più utenti diversi) e più sarà probabile fornire suggerimenti che si riveleranno essere corretti. La parola "collaborativo" quindi sta a indicare che i suggerimenti sono generati in base alle scelte degli altri, e non in base a ciò che ha strettamente scelto l'utente sul quale viene applicato il filtering. Difatti, anche se è A guardasse solo film horror, un recommender system collaborativo potrebbe suggerire anche titoli di generi differenti. A nota a margine, un filtro differente, basato sui gusti esclusivi dell'utente è conosciuto come *content-based filtering*.

L'altra componente della nostra soluzione è la **random walk**. Si tratta di un concetto matematico di tipo statistico, che descrive una serie di "passi" all'interno di uno spazio N-dimensionale. Un esempio banale di random walk: immaginiamo di trovarci sulla linea dei numeri interi (quindi in un ambiente bidimensionale), precisamente in posizione 0. una possibile random walk in questa situazione, consiste nello spostarsi nella linea in posizione $+1/-1$ (con probabilità 50 e 50). Altri esempi più complessi di possibili applicazioni si possono trovare in differenti ambiti, fra i quali: informatica, fisica, chimica, biologia, psicologia, economia. Esempio in Figura 2.1

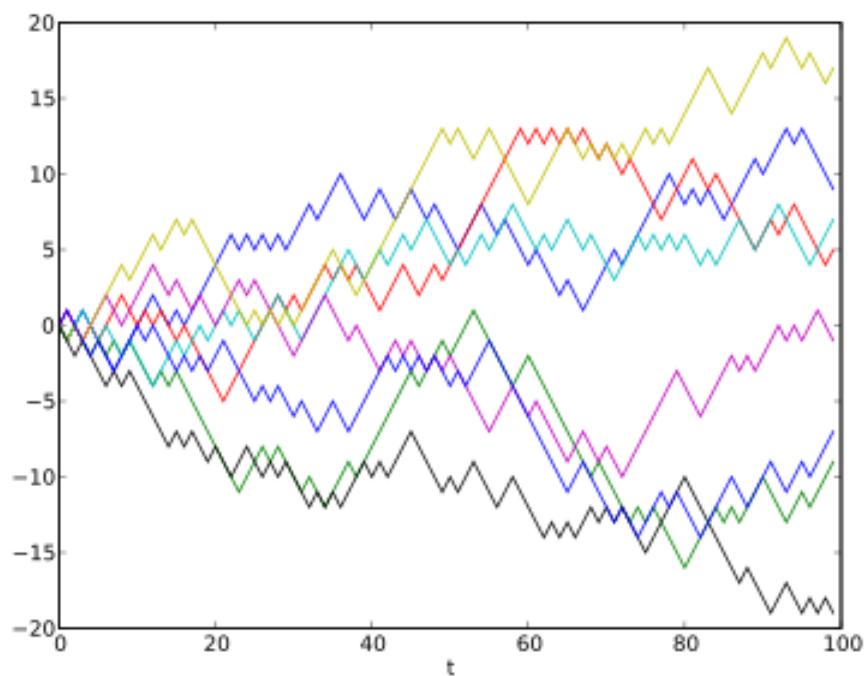


Figura 2.1: Schematizzazione dell'esempio sopra riportato. In figura vi sono 8 possibili random walk

fig:rwes

Nel nostro caso, la random walk andrà eseguita su di un grafo G non orientato, nel quale l'insieme dei nodi V è costituito sia dai film nel dataset, che dagli utenti, mentre nell'insieme degli archi E troviamo un collegamento fra due nodi nel caso che un utente abbia visto un certo film, e viceversa.

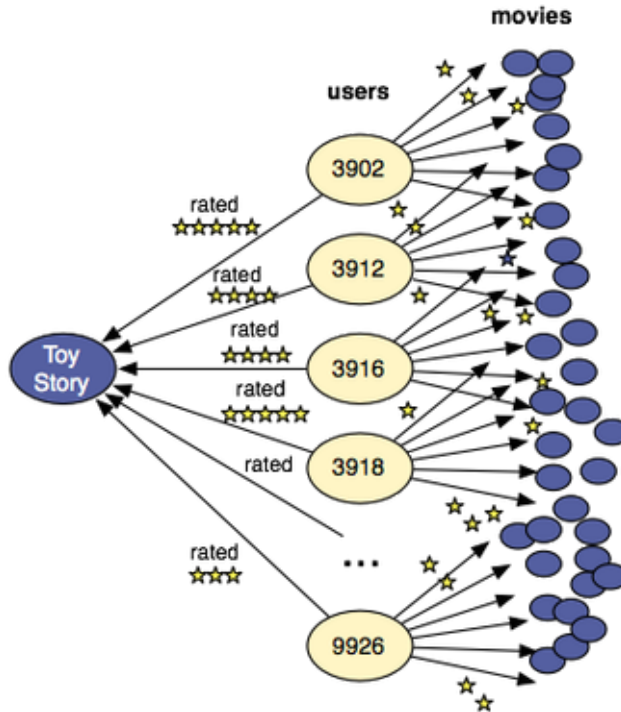


Figura 2.2: Porzione del grafo generato dal dataset. Notare che nel nostro caso sarà non orientato

fig:grasm

In un grafo generico, la random walk viene eseguita, intuitivamente, spostandosi da un nodo all'altro, tramite gli archi presenti. La probabilità di spostarsi da un nodo a un suo determinato vicino dipende dal grado del nodo stesso: più è grande e più sarà "difficile" (la probabilità di spostarsi su un vicino scelto è di $1/\text{deg}(\text{nodo})$). Se vogliamo rendere più "semplice" spostarsi verso un certo vertice, occorre quindi assegnare probabilità diverse ad archi diversi. Nel caso dei film, l'idea è quella di assegnare una maggiore probabilità se il voto del film è alto. In questo modo, si tenderà ad attraversare solo gli archi che portano ai film più acclamati, raggiungendo quindi risultati che potrebbero essere valutati positivamente dal nostro utente.

Capitolo 3

Soluzione proposta

Per semplicità è stata scelta la versione compatta del dataset Movielens, la quale contiene circa 100000 voti, suddivisi in 9000 diversi film da 600 utenti: in totale quindi un grafo con più di 10000 nodi e 100000 archi. Il programma è stato scritto in linguaggio **Python**¹ e si appoggia sulle seguenti librerie:

- **networkX**: la libreria "principale", tramite la quale è possibile eseguire operazioni su grafi ad alto livello, quindi in modo più semplice e libero.
- **pandas**: permette di lavorare facilmente con dataset, importandoli sottoforma di file csv e prepararli con il preprocessing del caso.
- **secret**: per la generazione sicura di valori casuali, che serviranno durante la random walk.

Per la creazione del grafo, è stato eseguito un *merge* con pandas di due dei csv originali di Movielens, in modo da avere subito tutti i dati necessari: userId, titolo del film e voto. Quest'ultimo è attributo dell'arco fra utente e film. In seguito, il dataframe così ottenuto viene diviso in due parti: train set e test set. Questo servirà per avere una stima della precisione dei suggerimenti che verranno calcolati.

Dopo avere quindi importato correttamente i grafi train e test, passiamo all'algoritmo per risolvere il nostro problema. Innanzitutto bisogna scegliere da quale utente partire, ovvero chi è che vuole ricevere i suggerimenti sui film? Pertanto, è stato aggiunto un nuovo utente al grafo (con id maggiore di 600, ad esempio "777") ed egli è stato collegato ai film presenti nel grafo che aveva visionato, con rispettivo voto.

```
1 | newUser = 777
  | G.add_edge(newUser, 'The Machinist (2004)', rating=5.0)
3 | G.add_edge(newUser, 'Harry Potter and the Prisoner of Azkaban (2004)',
  |   rating=4.0)
  | G.add_edge(newUser, 'Toy Story (1995)', rating=5.0)
5 | G.add_edge(newUser, 'Pulp Fiction (1994)', rating=5.0)
  | G.add_edge(newUser, 'The Mask', rating=4.0)
```

Aggiunta nuovo utente.

¹E' richiesta almeno la versione 3.5

3. Soluzione proposta

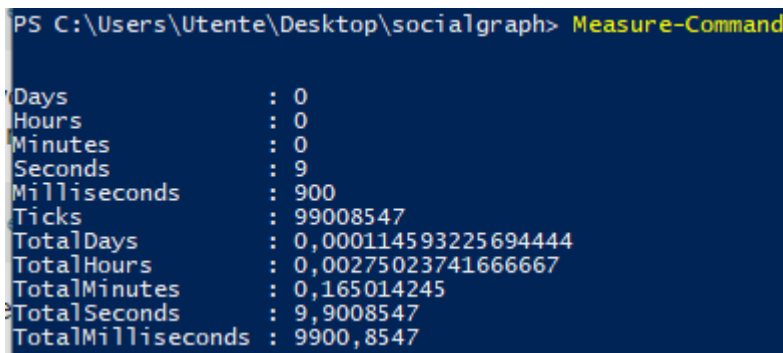
A questo punto la strategia da seguire è:

1. Si estrae a caso un film visto dall'utente (quindi un suo vicino), privilegiando la probabilità di quelli con voto alto.
2. Si estrae a caso un utente che ha visto e apprezzato quel film, anche qui privilegiando voti elevati.
3. Si estrae nuovamente un film visto dall'ultimo utente...
4. Il film selezionato viene temporaneamente salvato, e un contatore a esso associato è incrementato di 1.

Bisogna ripetere questo procedimento per un certo numero di volte², assicurandosi di non estrarre film già visti dal nostro utente (non ha senso raccomandare qualcosa che non è nuovo). Insomma, quindi, si cerca di "allontanarsi" dal nostro utente di partenza, passando per gli archi migliori, in termini di voto. L'ideale sarebbe evitare di allontanarsi eccessivamente dalla nostra radice, in quanto i suggerimenti tenderanno a perdere di efficacia. La lunghezza della random walk è stata settata a 3 "hop", valore ottimale per prevenire il degradamento delle previsioni.

Come fare a privilegiare un arco rispetto a un altro (leggasi: dirigersi verso i film con voto alto anziché basso)? Nel codice, la scelta è stata quella di ordinare gli archi per voto, e di prenderne il sottoinsieme di quelli più elevati, in base a una certa soglia (*threshold*) impostata a 0.75, corrispondente quindi alla top 25% dei film guardati. Per una scelta ancor più ristretta basta aumentare la soglia.

L'esecuzione del codice è relativamente veloce, su un portatile di fascia media del 2014 con Windows 10³ impiega poco meno di 10 secondi, come mostra la Figura 3.3



```
PS C:\Users\Utente\Desktop\socialgraph> Measure-Command
Days           : 0
Hours          : 0
Minutes        : 0
Seconds        : 9
Milliseconds   : 900
Ticks          : 99008547
TotalDays      : 0,000114593225694444
TotalHours     : 0,00275023741666667
TotalMinutes   : 0,165014245
TotalSeconds   : 9,9008547
TotalMilliseconds : 9900,8547
```

Figura 3.1: Tempo di esecuzione del programma.

fig:time

²Nel codice questo valore è impostato a 10000.

³processore Intel Core I5-4120U, 8GB RAM e scheda video integrata Intel 4400

Di seguito, un paio di esempi di possibili output.

```
Accuracy: 0.4
-----
Titoli suggeriti
Star Wars: Episode VI - Return of the Jedi (1983)
Aliens (1986)
Lord of the Rings: The Two Towers, The (2002)
Matrix, The (1999)
Lord of the Rings: The Fellowship of the Ring, The (2001)
>>>
```

Figura 3.2: Primo esempio di output.

fig:time

```
Accuracy: 0.36
-----
Titoli suggeriti (in ordine di attinenza
Matrix, The (1999)
Lord of the Rings: The Fellowship of the Ring, The (2001)
Star Wars: Episode IV - A New Hope (1977)
Forrest Gump (1994)
Seven (a.k.a. Se7en) (1995)
>>>
```

Figura 3.3: Secondo esempio di output.

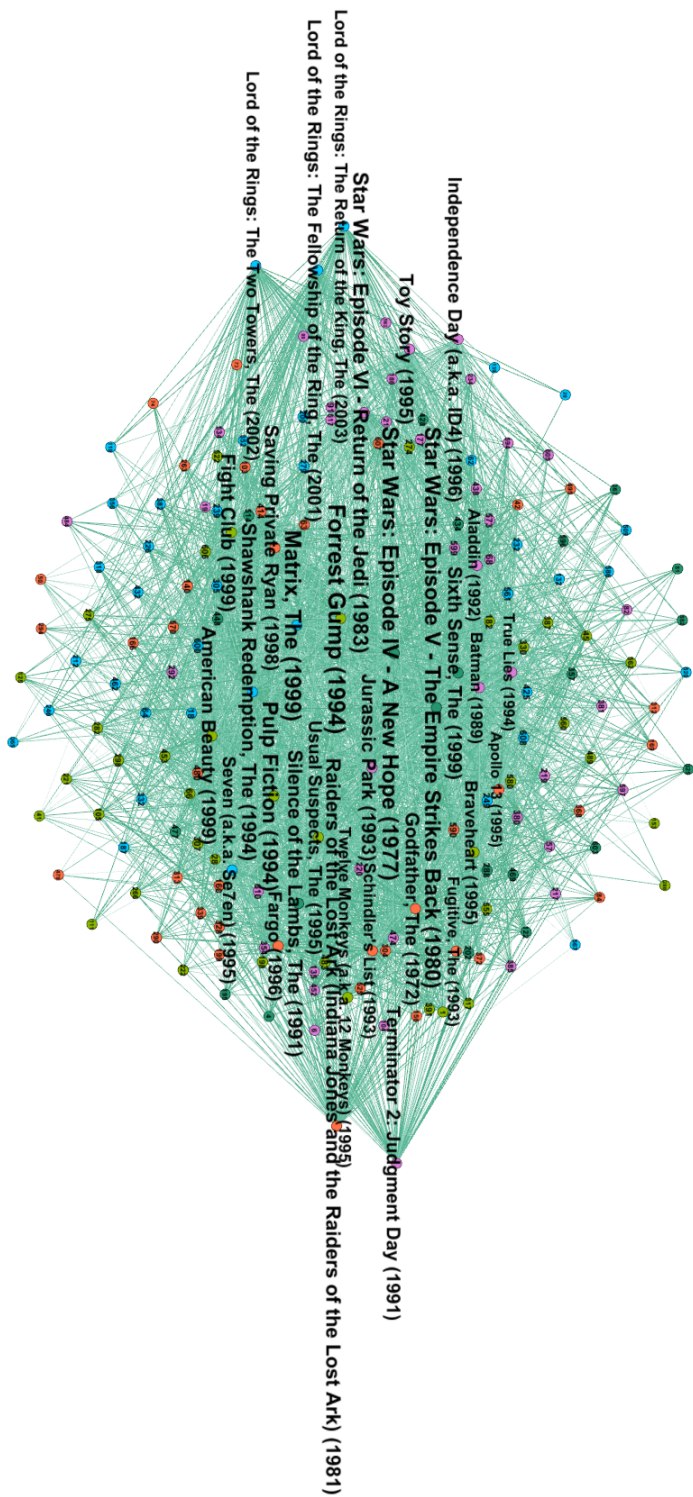
fig:time

Il programma elabora due differenti liste di film, le quali poi vengono confrontate fra loro per verificare la validità delle predizioni calcolate. I film che appaiono in entrambe le liste, sono i candidati designati da mostrare nell'output del programma. Verranno poi stampati a schermo i 5 migliori film, in ordine di preferenza secondo quanto rilevato durante le random walk. Il valore di precisione oscilla fra 0.3 e 0.4, ciò dipende principalmente dal modo in cui è stata effettuata la divisione del dataframe originale in train e test set.

Alla pagina seguente, si trova una visualizzazione di parte del grafo creato dal dataset originale, precisamente possiamo vedere i nodi con il grado più alto, ovvero chi sono gli utenti più "esperti"⁴ e, soprattutto, quali sono i film più visti. I nodi di colore diverso indicano differenti comunità, anche se si tratta di una divisione non molto chiara, in quanto appaiono allineati anche film con genere piuttosto diverso l'uno dall'altro. La Figura è stata elaborata tramite il software di analisi di *social graph* **Gephi**.

⁴Abbastanza inutile visualizzare dati a riguardo, in quanto è disponibile solo il loro id numerico.

10



Capitolo 4

Conclusioni

In questo report è stato mostrato un approccio, basato sull'esplorazione del grafo generato da un dataset, ai sistemi di raccomandazione. In particolare, la scelta è caduta sulla tecnica statistica della random walk, la quale ha permesso di raggiungere buoni risultati, più o meno coerenti con quanto dato in input. La forza di questo metodo è che permette di ricevere consigli non banali, nel senso che non sono strettamente legati al genere preferito dall'utente in cerca di consigli, ma possono trattarsi quindi di film all'apparenza lontani dal suo gusto. Un punto debole della tecnica potrebbe essere in una sua ipotetica gestione troppo permissiva¹: in questo caso si rischierebbe davvero di ottenere suggerimenti slegati dalla lista dell'utente.

In appendice vi è l'intero codice Python, con le istruzioni per poterlo eseguire e replicare quindi i risultati qui mostrati in precedenza.

¹Nel caso del nostro codice, la camminata ha un limite fisso, relativamente basso.

Appendice A

Appendice

- Scaricare il codice e i file csv dalla *repository Github*¹ in una cartella del proprio PC.
- Spostarsi nel path corretto da terminale con l'istruzione *cd*.
- Lanciare il programma con *python MovieRandomWalk.py*.

Volendo, si può aggiungere il flag *"-i"* per poter reperire informazioni aggiuntive, come visualizzare gli utenti che hanno aiutato ad arrivare ai suggerimenti.

```
import networkx as nx
import numpy as np
import pandas as pd
4 #from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
6 import secrets as sc #per la funzione random
from collections import Counter

8
#funzione per estrarre il prossimo nodo in modo casuale, con la probabilità
  condizionata dal rating
10 #la variabile t serve a limitare il sottoinsieme di nodi da estrarre: più il
    valore è alto e più nodi saranno filtrati
def extract_neigh(node, t = 0.75):
12     neig = sorted(node.items(), key=lambda edge: edge[1]['rating'])
    threshold = int(len(neig)*t)
14     neig = neig[threshold:]
    next = sc.choice(neig)
16     return next[0]

18 #set dei tipi delle colonne del dataset per una lettura più agevole
typesMovies = {'movieId': np.dtype(int), 'title': np.dtype(str), 'genres' : np
    .dtype(str)}
20 typesRatings = {'userId': np.dtype(int), 'movieId': np.dtype(int), 'rating':
    np.dtype(float), 'timestamp' : np.dtype(int)}

22 #import dei dataset originali
movies = pd.read_csv('movies.csv', dtype=typesMovies, low_memory = False)
24 ratings = pd.read_csv('ratings.csv', dtype=typesRatings, low_memory = False)

26 #unione dei due dataset
movie_ratings = pd.merge(movies, ratings)
28 #drop delle colonne inutili
movie_ratings.drop("timestamp", inplace = True, axis = 1)
30 movie_ratings.drop("movieId", inplace = True, axis = 1)
movie_ratings.drop("genres", inplace = True, axis = 1)
```

¹<https://github.com/G0D0T/MovieRandomWalk>

A. Appendice

```
32 movie_ratings.to_csv('import.csv', index=False)
34 #train, test = train_test_split(movie_ratings, test_size=0.5)
35 #TRAIN=0.75/TEST=0.25
36 kf = KFold(n_splits = 4, shuffle = True)#, random_state = 0)
37 result = next(kf.split(movie_ratings), None)
38 train = movie_ratings.iloc[result[0]]
39 test = movie_ratings.iloc[result[1]]
40 print('Numero righe del train set: ',len(train))
41 print('Numero righe del test set: ',len(test))
42 #print(movie_ratings.sample(10))
43 print('-----')
44
45 #generazione del grafo train
46 Gtrain = nx.from_pandas_edgelist(train, 'userId', 'title', ['rating'])
47 Gtrain.name = 'Train_Set'
48 print(nx.info(Gtrain))
49 print('-----')
50 #generazione del grafo test
51 Gtest = nx.from_pandas_edgelist(test, 'userId', 'title', ['rating'])
52 Gtest.name = 'Test_Set'
53 print(nx.info(Gtest))
54 #nx.write_weighted_edgelist(G, 'movie.weighted.edgelist')
55 print('-----')
56
57 #aggiungo un nuovo utente con relativi film visti al train set
58 newUser = 777
59 Gtrain.add_edge(newUser, 'The Machinist (2004)', rating=3.0)
60 Gtrain.add_edge(newUser, 'Harry Potter and the Prisoner of Azkaban (2004)',
61                  rating=4.0)
62 Gtrain.add_edge(newUser, 'Toy Story (1995)', rating=5.0)
63 Gtrain.add_edge(newUser, 'Pulp Fiction (1994)', rating=5.0)
64 Gtrain.add_edge(newUser, 'The Mask', rating=4.0)
65 Gtrain.add_edge(newUser, 'Raiders of the Lost Ark (Indiana Jones and the
66                  Raiders of the Lost Ark) (1981)', rating=4.5)
67 Gtrain.add_edge(newUser, 'Shawshank Redemption, The (1994)', rating=5.0)
68 Gtrain.add_edge(newUser, 'Mission: Impossible - Rogue Nation (2015)', rating
69                  =2.0)
70 Gtrain.add_edge(newUser, 'Back to the Future (1985)', rating=4.5)
71
72 #idem per il test set
73 Gtest.add_edge(newUser, 'The Machinist (2004)', rating=3.0)
74 Gtest.add_edge(newUser, 'Harry Potter and the Prisoner of Azkaban (2004)',
75                 rating=4.0)
76 Gtest.add_edge(newUser, 'Toy Story (1995)', rating=5.0)
77 Gtest.add_edge(newUser, 'Pulp Fiction (1994)', rating=5.0)
78 Gtest.add_edge(newUser, 'The Mask', rating=4.0)
79 Gtest.add_edge(newUser, 'Raiders of the Lost Ark (Indiana Jones and the
80                 Raiders of the Lost Ark) (1981)', rating=4.5)
81 Gtest.add_edge(newUser, 'Shawshank Redemption, The (1994)', rating=5.0)
82 Gtest.add_edge(newUser, 'Mission: Impossible - Rogue Nation (2015)', rating
83                 =2.0)
84 Gtest.add_edge(newUser, 'Back to the Future (1985)', rating=4.5)
85
86 #parametri globali
87 ripetizioni = 10000
88 numSteps = 2 #lunghezza prevista della random walk
89 top = 25
```

```

walk = []
84 visitedFilm = []
   visitedUser = []
86 start = 777 #da sostituire con l'id utente che vuole avere un consiglio

88 def run (grafo):
   giavisti = ([n for n in grafo.neighbors(start)])
90   step = 0
   flag = False #True per gli utenti, viceversa per i film
92   visitedUser.append(start)
   for f in giavisti:
94     visitedFilm.append(f) #salvo i film già visionati per evitare di
       consigliarli di nuovo
   nodo = start
96   #prev = start
   primoF = True #flag che indica se ci troviamo ad aggiungere il primo film

98   for i in range(0,ripetizioni):
100     while (step < numSteps):
       nodo = extract_neigh(grafo[nodo])
102       #print(nodo)
       #print(prev)
104       if flag:
         if nodo not in visitedUser:
106           visitedUser.append(nodo)
         else:      #nel caso il nodo sia già stato visitato, passo alla
           prossima iterazione
108           nodo = prev
           continue
110       else:
         if nodo not in visitedFilm:
112           visitedFilm.append(nodo)
         elif primoF:
114           primoF = False
           flag = not flag
116           prev = nodo
           continue
118       else:
         nodo = prev
120         continue

122     #walk.append(nodo) #il nodo trovato in questa iterazione è nuovo,
       quindi devo aggiornare i seguenti parametri
       prev = nodo
124     flag = not flag
       step += 1
126     #print('-----')

128     #Salvataggio del film trovato con l'ultima walk
       walk.append(visitedFilm[-1])
130     #reset dei vari parametri
       nodo = start
132     primoF = True
       flag = False
134     step = 0
       visitedUser.clear()
136     visitedUser.append(start)

```

```

138     visitedFilm.clear()
139     for f in giavisti:
140         visitedFilm.append(f)
141
142     #conta delle occorrenze dei film durante le ripetizioni della random walk
143     + ordinamento
144     res = sorted(Counter(walk).items(), key=lambda x: x[1], reverse=True)
145     for r in res[:top]:
146         print(r)
147
148     return res[:top]
149
150 print('Consigli TRAIN set')
151 trainlista = run(Gtrain)
152 walk.clear()
153 visitedUser.clear()
154 visitedFilm.clear()
155 print('-----')
156 print('Consigli TEST set')
157 print('-----')
158 testlista = run(Gtest)
159
160 #dopo aver eseguito le walk sia su train che test, si stima la precisione
161 dei suggerimenti
162 conta = 0
163 common = []
164 for i in range(0,len(trainlista)):
165     for j in range(0,len(testlista)):
166         if trainlista[i][0] == testlista[j][0]:
167             common.append(testlista[j])
168             conta +=1
169
170 #controllo dei film suggeriti in entrambi i set e stampa a schermo la top 5
171 conta /= top
172 print("\n\nAccuracy: ", conta)
173 print('-----')
174 print("\nTitoli suggeriti (in ordine di attinenza)")
175 common.sort(key=lambda x: x[1], reverse=True)
176 for elem in common[:5]:
177     print (elem[0])
178
179 """
180 funzioni utils
181 #print([n for n in G.neighbors(2)]) #vedo tutti i film visti da un certo
182 utente
183 #print(G[2]) #vedo tutti i film visti da un certo utente con voto assegnato
184 #print([n for n in G.neighbors('Whiplash (2014)')]) #vedo tutti gli utenti
185 che hanno visto un certo film
186 for x in G.nodes():
187     print ("Node:", x, "has total #degree:",G.degree(x), " , In_degree: ",
188           G.out_degree(x)," and out_degree: ", G.in_degree(x))
189     for u,v in G.edges():
190         print ("Weight of Edge (" +str(u)+"," +str(v)+")", G.get_edge_data(u,v))
191
192 Data = open('ratings.csv', "r")
193 next(Data, None) # skip the first line in the input file
194 Graphtype = nx.Graph() #undirected

```

```
190 | G = nx.parse_edgelist(Data, delimiter=';', create_using=GraphType, nodetype=  
    | int, data= (('rating', float),))  
    | """
```

MovieRandomWalk.py