



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	13
A Appendice	15

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

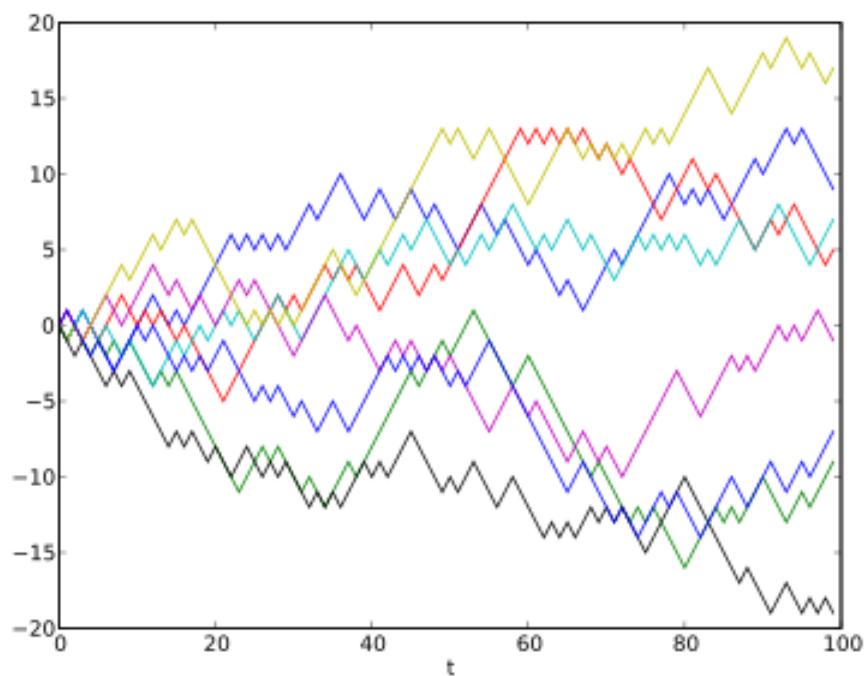


Figura 2.1: Schematizzazione dell'esempio sopra riportato. In figura vi sono 8 possibili random walk

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.

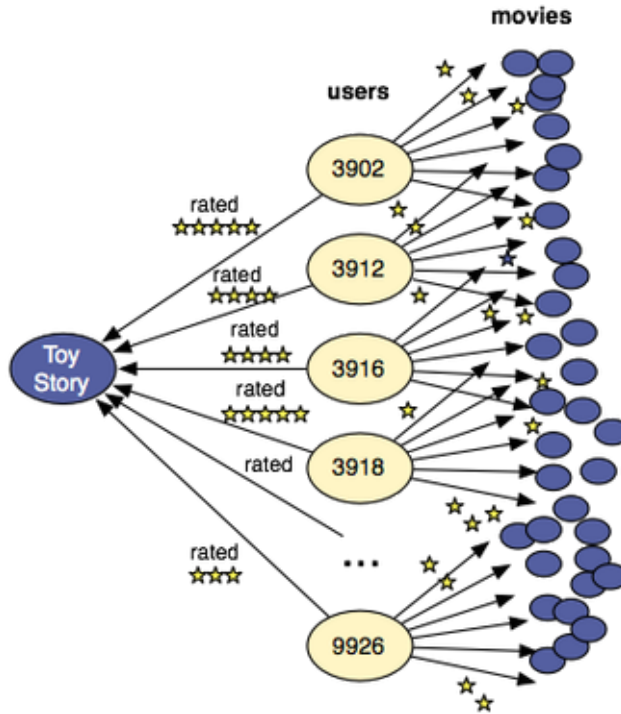


Figura 2.2: Porzione del grafo generato dal dataset. Notare che nel nostro caso sarà non orientato

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 il grafo per il train, passiamo all'algoritmo per risolvere il nostro problema. Ogni riga del test set è un candidato utente, quindi, uno alla volta, si proverà a verificare se quella data raccomandazione può essere prevista dal recommender system dal grafo costruito sul train set. Dopo una ricerca esaustiva, verrà calcolata la precisione delle previsioni generate.

```
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

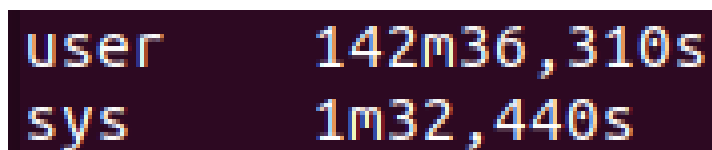
A questo punto la strategia da seguire è:

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 piuttosto lenta, su un portatile di fascia media del 2014 con Ubuntu 18.04³ impiega più di 2 ore (intorno a 140 minuti), come mostra la Figura 3.3



```
user      142m36,310s
sys       1m32,440s
```

Figura 3.1: Tempo di esecuzione del programma.

fig:time

²Nel codice questo valore è impostato a 1000.

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

Di seguito, un paio di esempi di possibili output, presi dall'interno del file generato con tutti i suggerimenti per ogni istanza del test set. Per brevità vengono mostrati solo i vari tipo di precisione:

- Precisione delle varie top: indica se il film era fra i più suggeriti (top5, 10 o 15).
- Hit: indica se il film cercato era nella lista ritornata dalla random walk (sia top che non).
- SuperHit: indica se il film cercato era il primo della lista.

```
Precisione delle top 5: 0.38579787761578895 3893
Precisione delle top 10: 0.23108201924030545 2334
Precisione delle top 15: 0.14975701676088466 1511
Hit: 4269
SuperHit: 50
```

Figura 3.2: Primo esempio di output.

fig:time

```
Precisione delle top 5: 0.36001190121987506 3631
Precisione delle top 10: 0.25785976395913915 2601
Precisione delle top 15: 0.18248537141723694 1840
Hit: 4421
SuperHit: 53
```

Figura 3.3: Secondo esempio di output.

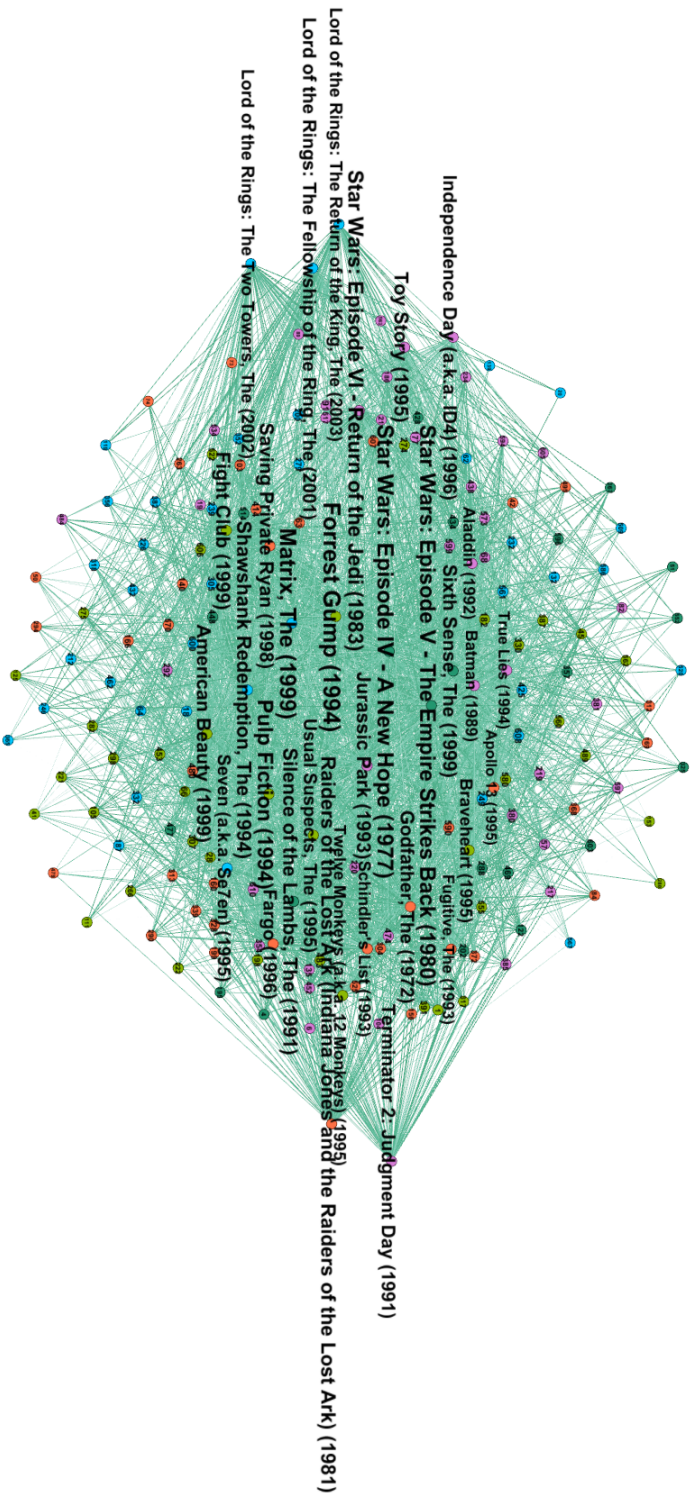
fig:time

Il programma, tramite il grafo del train set, prova a ricostruire le previsioni contenute nel test set. Sia in caso positivo che negativo, per ogni utente è stampata in un file una top 15 dei film più suggeriti per quella data istanza, in base al percorso della random walk. Il numero di Hit supera il 40%, mentre le SuperHit sono molto più rare (non arrivano all'1%) ciò potrebbe dipendere dal modo in cui è stata effettuata la divisione del dataframe originale in train e test set, ma anche al fatto che alcuni film sono generalmente più probabili di altri in fatto di raccomandazioni, in quanto maggiormente presenti e con voti alti.

3. Soluzione proposta

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.



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 addizionali, come visualizzare gli utenti che hanno aiutato ad arrivare ai suggerimenti.

```
import networkx as nx #per la gestione del grafo
2 import numpy as np
import pandas as pd
4 from sklearn.model_selection import KFold #per lo split del train e test set
import secrets as sc #per la funzione random
6 from collections import Counter

8 #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)}
10 typesRatings = {'userId': np.dtype(int), 'movieId': np.dtype(int), 'rating':
np.dtype(float), 'timestamp': np.dtype(int)}

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

16 #unione dei due dataset
movie_ratings = pd.merge(movies, ratings)
18 #drop delle colonne inutili
movie_ratings.drop("timestamp", inplace = True, axis = 1)
20 movie_ratings.drop("movieId", inplace = True, axis = 1)
movie_ratings.drop("genres", inplace = True, axis = 1)
22 #movie_ratings.to_csv('import.csv', index=False)

24 #train, test = train_test_split(movie_ratings, test_size=0.5)
#TRAIN=0.9/TEST=0.1
26 kf = KFold(n_splits = 10, shuffle = True)#, random_state = 0)
result = next(kf.split(movie_ratings), None)
28 train = movie_ratings.iloc[result[0]]
test = movie_ratings.iloc[result[1]]
30 print('Numero righe del train set: ', len(train))
print('Numero righe del test set: ', len(test))
32 #print(movie_ratings.sample(10))
print('-----')
```

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

```
34 | #generazione del grafo train
36 | Grafo = nx.from_pandas_edgelist(train, 'userId', 'title', ['rating'])
    | Grafo.name = 'Train_Set'
38 | print(nx.info(Grafo))
    | print('-----')
40 | print('ATTENDERE...')

42 | #variabili e contatori vari
    | ripetere = 1000
44 | passi = 3
    | hit = 0
46 | hit5 = 0
    | hit10 = 0
48 | hit15 = 0
    | superhit = 0
50 | top = 5
    | visitatiF = []
52 | visitatiU = []
    | walk = []
54 |
    | temp = 0
56 |
    | #funzione che estrae il prossimo "hop" della random walk
58 | def extract_neigh(node, t = 0.75):
    |     neig = sorted(node.items(), key=lambda edge: edge[1]['rating'])
60 |     threshold = int(len(neig)*t)
    |     neig = neig[threshold:]
62 |     next = sc.choice(neig)
    |     return next[0]
64 |
    | #funzione che ritorna il "traguardo" della random walk
66 | def randomwalk(start):
    |     visti = ([n for n in Grafo.neighbors(start)])
68 |     for f in visti:
    |         visitatiF.append(f)
70 |     visitatiU.append(start)
    |     nodo = start
72 |     step = 0
    |     tempc = 0
74 |     tempcc = 0
    |     while step < passi:
76 |         nodo = extract_neigh(Grafo[nodo])
    |         #se il nodo ha valore int o meno viene gestito come userId o film
78 |         if (isinstance(nodo, int)):
    |             if nodo in visitatiU:
80 |                 if tempc > 7:
    |                     break
82 |                 tempc += 1
    |                 nodo = prev
84 |                 continue
    |             else:
86 |                 visitatiU.append(nodo)
    |         else:
88 |             if step == 0:
    |                 pass#do nothing
90 |             elif nodo in visitatiF:
```

```

    if tempcc > 7:
        break
    tempcc += 1
    nodo = prev
    continue
else:
    visitatiF.append(nodo)

step += 1
prev = nodo

res = visitatiF[-1]
visitatiF.clear()
visitatiU.clear()
return res

for index, row in test.iterrows():
    temp += 1
    user = row['userId']
    film = row['title']
    for i in range(0, ripetere):
        r = randomwalk(user)
        walk.append(r)
    #conta delle occorrenze dei film durante le ripetizioni della random walk
    + ordinamento
    consigli = sorted(Counter(walk).items(), key=lambda x: x[1], reverse=True)
    walk.clear()
    #controllo del film in lista:
    #SuperHit = primo della lista -> stesso film del test set
    #Top = il film è in alto nella lista dei suggerimenti
    #Hit = il film è nella lista dei suggerimenti
    if film == consigli[0][0]:
        superhit += 1
    for i in range(0, len(consigli)):
        if film in consigli[i][0]:
            hit += 1
            if film in consigli[i][0] and i < top:
                hit5 += 1
            elif film in consigli[i][0] and i < top+5:
                hit10 += 1
            elif film in consigli[i][0] and i < top+10:
                hit15 += 1
    #la top 15 dei suggerimenti viene stampata all'interno di un file
    with open("o.txt", "a") as f:
        print("Film suggeriti per l'utente: ", user, film, file=f)
        for elem in consigli[:top+10]:
            print(elem, file=f)
    #stampa a schermo dei vari contatori a ogni iterazione
    print(temp, hit, hit5, hit10, hit15, superhit)

#calcolo della precisione dei suggerimenti e stampa (su file)
tot = len(test.index)
accuracy5 = hit5/tot
accuracy10 = hit10/tot
accuracy15 = hit15/tot
print('-----')
with open("o.txt", "a") as f:

```

```
148 | print("RISULTATO FINALE...\n\n")
    | print('Hit: ',hit, file=f)
    |     print('Precisione delle top 5: ',accuracy5, hit5, file=f)
150 |     print('Precisione delle top 10: ',accuracy10, hit10, file=f)
    |     print('Precisione delle top 15: ',accuracy15, hit15, file=f)
152 |     print('SuperHit: ', superhit, file=f)
```

MovieRandomWalk.py