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

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

Indice

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 cinematografiche[1].
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, September 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

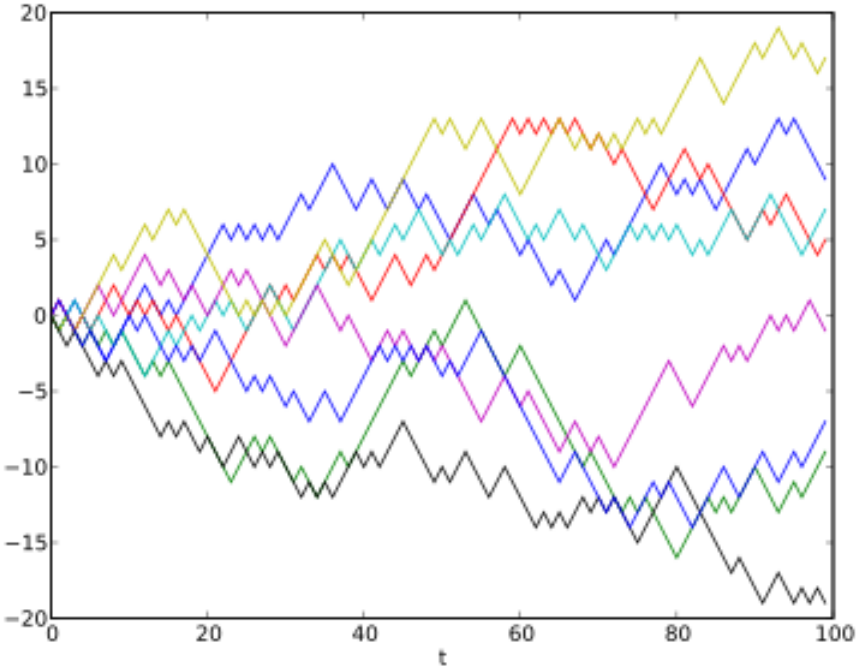


Figura 2.1: Schematizzazione dell'esempio sopra riportato. In figura vi sono 8 possibili random walk

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.

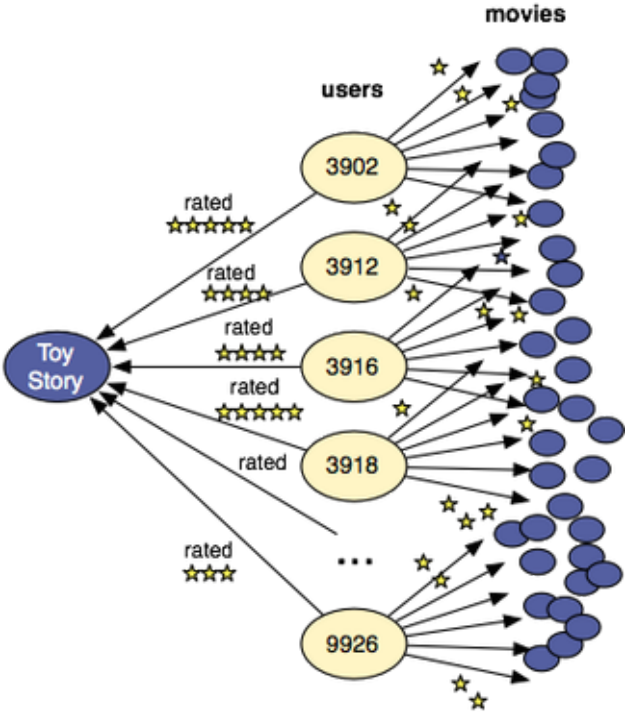


Figura 2.2: Porzione del grafo generato dal dataset. Notare che nel nostro caso sarà non orientato

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 $1/deg(nodo)$). 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.
- *numpy*: libreria matematica.
- *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.

Dopo avere quindi importato correttamente il grafo, 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.

```
newUser = 777
G.add_edge(newUser, 'The Machinist (2004)', rating=5.0)
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)
G.add_edge(newUser, 'Pulp Fiction (1994)', rating=5.0)
G.add_edge(newUser, 'The Mask', rating=4.0)
```

Aggiunta nuovo utente.

[1] E' richiesta almeno la versione 3.5

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. Etc...

Bisogna ripetere questo procedimento per un certo numero di volte, assicurandosi di non estrarre film già visti dal nostro utente (non ha senso raccomandare qualcosa che non è nuovo) e anche utenti diversi. 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.

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 10 impiega poco meno di 3 secondi, come mostra la Figura 3.3

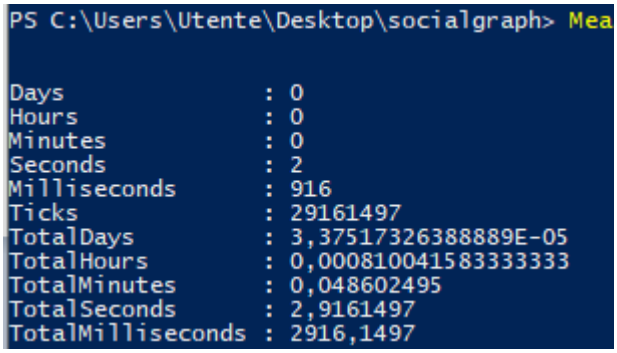


Figura 3.1: Tempo di esecuzione del programma. fig:time

Di seguito, un paio di esempi di possibili output.

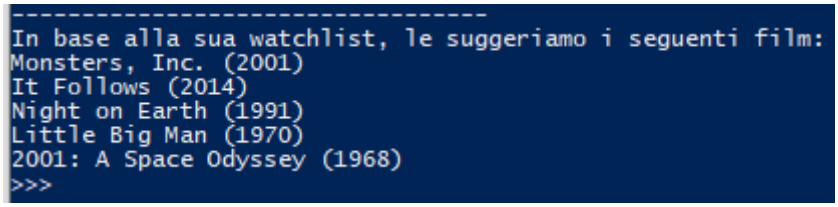


Figura 3.2: Primo esempio di output. fig:time

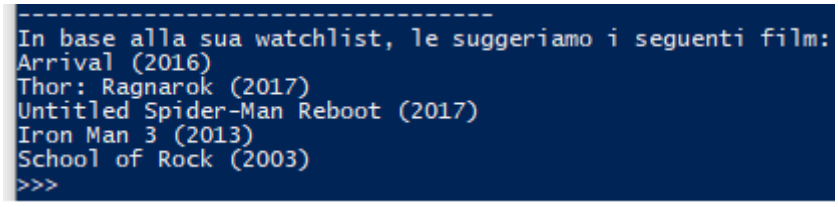
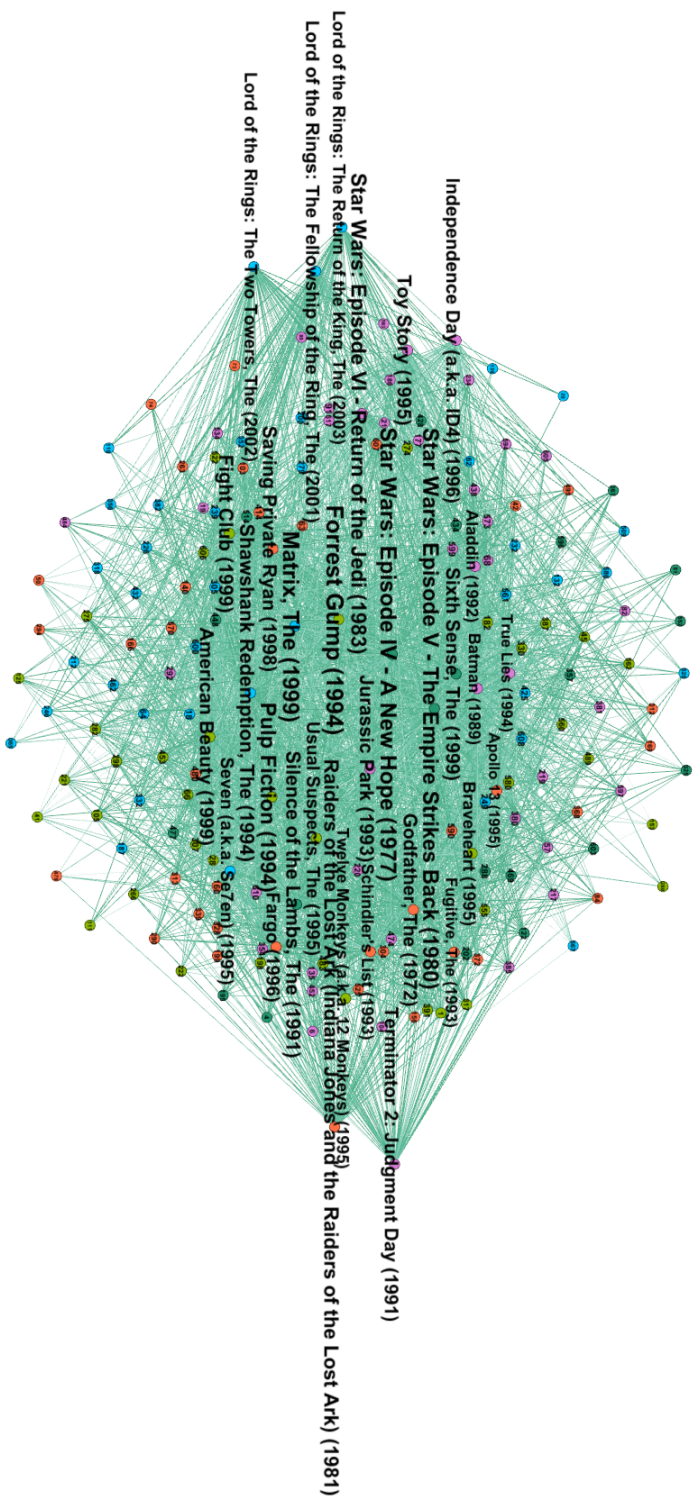


Figura 3.3: Secondo esempio di output. fig:time

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"[2] 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**.

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


Independence Day (a.k.a. ID4) (1996)
Star Wars: Episode V - The Empire Strikes Back (1980)
Toy Story (1995)
Star Wars: Episode IV - A New Hope (1977)
Star Wars: Episode VI - Return of the Jedi (1983)
Lord of the Rings: The Return of the King, The (2003)
Lord of the Rings: The Fellowship of the Ring, The (2001)
Forrest Gump (1994)
Matrix, The (1999)
Saving Private Ryan (1998)
Lord of the Rings: The Two Towers, The (2002)
Fight Club (1999)
Shawshank Redemption, The (1994)
American Beauty (1999)
Seven (a.k.a. Se7en) (1995)
Pulp Fiction (1994)
Fargo (1996)
Silence of the Lambs, The (1991)
Usual Suspects, The (1995)
Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)
Twelve Monkeys (a.k.a. 12 Monkeys) (1995)
Schindler's List (1993)
Jurassic Park (1993)
Godfather, The (1972)
Terminator 2: Judgment Day (1991)
Sixth Sense, The (1999)
Braveheart (1995)
Fugitive, The (1993)
Aladdin (1992)
Batman (1989)
True Lies (1994)
Apollo 13 (1995)

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 permissiva[1]: 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.

[1] Nel 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
import numpy as np
import pandas as pd
import secrets as sc #per la funzione random

#funzione per estrarre il prossimo nodo in modo casuale, con la probabilità condizionata dal rating
#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):
  neig = sorted(node.items(), key=lambda edge: edge[1]['rating'])
  threshold = int(len(neig)*t)
  neig = neig[threshold:]
  next = sc.choice(neig)
  return next[0]

#funzione finale per la stampa a schermo di quanto trovato
def consigli(reccomend, filt):
  print('---------------------------------')
  print('In base alla sua watchlist, le suggeriamo i seguenti film: ')
  for f in reccomend[filt:]:
    print(f)

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

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

#unione dei due dataset
movie_ratings = pd.merge(movies, ratings)
```

[1] https://github.com/G0D0T/MovieRandomWalk

```
32
#drop delle colonne inutili
34 movie_ratings.drop("timestamp", inplace = True, axis = 1)
movie_ratings.drop("movieId", inplace = True, axis = 1)
36 movie_ratings.drop("genres", inplace = True, axis = 1)

38 #print(movie_ratings.sample(10))

40 #generazione del grafo
G = nx.from_pandas_edgelist(movie_ratings, 'userId', 'title', ['rating'])
42 movie_ratings.to_csv('import.csv', index=False)
print(nx.info(G))
44 #nx.write_weighted_edgelist(G, 'movie.weighted.edgelist')

46 #aggiungo un nuovo utente con relativi film visti
newUser = 777
48 G.add_edge(newUser, 'The Machinist (2004)', rating=5.0)
G.add_edge(newUser, 'Harry Potter and the Prisoner of Azkaban (2004)',
    rating=4.0)
50 G.add_edge(newUser, 'Toy Story (1995)', rating=5.0)
G.add_edge(newUser, 'Pulp Fiction (1994)', rating=5.0)
52 G.add_edge(newUser, 'The Mask', rating=4.0)

54 #print([n for n in G.neighbors(2)]) #vedo tutti i film visti da un certo
    utente
#print(G[2]) #vedo tutti i film visti da un certo utente con voto assegnato
56 #print([n for n in G.neighbors('Whiplash (2014)')]) #vedo tutti gli utenti
    che hanno visto un certo film

58 #parametri globali
numSteps = 10 #lunghezza prevista della random walk
60 walk = []
visitedFilm = []
62 visitedUser = []
start = 777 #da sostituire con l'id utente che vuole avere un consiglio
64 giavisti = ([n for n in G.neighbors(start)])#.items()

66 def main():
  step = 0
68   flag = False #True per gli utenti, viceversa per i film
  walk.append(start)
70   visitedUser.append(start)
  for f in giavisti:
72     visitedFilm.append(f) #salvo i film già visionati per evitare di
     consigliarli di nuovo
  nodo = start
74   #prev = start
  primoF = True #flag che indica se ci troviamo ad aggiungere il primo film
76   endd = 0 #variabile che evita loop infiniti (possibile in un certo caso)

78   while (step < numSteps) and (endd < 30):
    if endd > 15: #verifico di non trovarmi in un "vicolo cieco" e, in caso,
      ritorno indietro
80       if flag:
          nodo = visitedUser[-2]
82         flag = not flag
       else:
```

```
84          nodo = visitedFilm[-2]
            flag = not flag
86      endd += 1
        nodo = extract_neigh(G[nodo])
88      print(nodo)
        #print(prev)
90      if flag:
          if nodo not in visitedUser:
92          visitedUser.append(nodo)
          else:     #nel caso il nodo sia già stato visitato, passo alla
         prossima iterazione
94          nodo = prev
            continue
96      else:
          if nodo not in visitedFilm:
98          visitedFilm.append(nodo)
          elif primoF:
100         primoF = False
            flag = not flag
102         prev = nodo
            continue
104       else:
            nodo = prev
106         continue

108     walk.append(nodo) #il nodo trovato in questa iterazione è nuovo, quindi
         devo aggiornare i seguenti parametri
        prev = nodo
110     flag = not flag
        step += 1
112
    consigli(visitedFilm, len(giavisti))
114
main()
116
"""
118 funzioni utils

120 for x in G.nodes():
          print ("Node:", x, "has total #degree:",G.degree(x), " , In_degree: ",
         G.out_degree(x)," and out_degree: ", G.in_degree(x))
122 for u,v in G.edges():
          print ("Weight of Edge ("+str(u)+","+str(v)+")", G.get_edge_data(u,v))
124


126     Data = open('ratings.csv', "r")
    next(Data, None)  # skip the first line in the input file
128 Graphtype = nx.Graph()  #undirected

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

MovieRandomWalk.py