Data Description: The dataset used for this project is the Netflix Movie Rating Dataset from Kaggle (Source: https://www.kaggle.com/netflix-inc/netflix-prize-data)

This dataset is large enough to build a good recommendation model and is adapted from 'Netflix Prize Dataset' which is so large that programmers may face memory issues while training a model using that dataset.

Movie file description: Movie_ID, Name, Year Rating file description: Movie_ID, User_ID, Name, Year

Let us build a Latent - Factor Based Recommender System to recommend movies to users. Part 1: Setting up the data:

In [1]:
```
from collections import defaultdict
import scipy
import scipy.optimize
import numpy
import random
```

In [2]:
```
path = "D:/TRL/OneDrive/CourseraPython/Final_Course4/NetflixRatingData/Netflix_Data
path2 = "D:/TRL/OneDrive/CourseraPython/Final_Course4/NetflixRatingData/Netflix_Dat
```

In [3]:
```
f = open(path)
```

In [4]:
```
header = f.readline()
```

In [5]:
```
header = header.strip().split(',')
```

In [6]:
```
header
```

Out[6]:
```
['User_ID', 'Rating', 'Movie_ID']
```

In [7]:
```
dataset = []
```

In [8]:
```
for line in f:
    fields = line.strip().split(',')
    d = dict(zip(header, fields))
    d['Rating'] = int(d['Rating'])
    d['User_ID'] = int(d['User_ID'])
    d['Movie_ID'] = int(d['Movie_ID'])
    dataset.append(d)
```

In [9]:
```
type(d['User_ID']), type(d['Movie_ID']), type(d['Rating'])
```

Out[9]:
```
(int, int, int)
```

In [10]:
```
dataset[0]
```

Out[10]:
```
{'User_ID': 712664, 'Rating': 5, 'Movie_ID': 3}
```

In [11]:
```
# create a dictionary with movie data, i.e. movie id, year of release
# and movie name
f2 = open(path2)
header = f2.readline()
header = header.strip().split(',')
```

```
movie_data = []
for line in f2:
    fields2 = line.strip().split(',')
    d = dict(zip(header, fields2))
    movie_data.append(d)
```

In [12]:
```
movie_data[0]
```
Out[12]:
```
{'Movie_ID': '1', 'Year': '2003', 'Name': 'Dinosaur Planet'}
```

In [13]:
```
movie_data[0]['Movie_ID']
```
Out[13]:
```
'1'
```

In [14]:
```
k = [item for item in movie_data if item['Movie_ID']=='1']
k
```
Out[14]:
```
[{'Movie_ID': '1', 'Year': '2003', 'Name': 'Dinosaur Planet'}]
```

In [15]:
```
for i in movie_data:
    if i['Movie_ID'] == '1':
        print(i)
        break
```
```
{'Movie_ID': '1', 'Year': '2003', 'Name': 'Dinosaur Planet'}
```

In [16]:
```
type(movie_data), type(movie_data[0])
```
Out[16]:
```
(list, dict)
```

Part 2: Finding Similarities Predict a rating to a new movie based on the ratings given to movies by a user and the movie names. We can also calculate the total votes and helpful votes for each movie.

In [17]:
```
# users per movie refer to the no. of Netflix users who watched and rated a movie
# movies per user refer to the no. of movies that were rated by netflix users.
UsersPerMovie = defaultdict(set)
MoviesPerUser = defaultdict(set)

MovieNames = {}

for d in dataset:
    user,movie = d['User_ID'], d['Movie_ID']
    UsersPerMovie[movie].add(user)
    MoviesPerUser[user].add(movie)
```

Functions to find Similarities:

Jaccard function calculates similarity between two movies: by dividing the number of users who watched both the movies by the number of users who watched either movie.

MostSimilar function : takes an input of a movie id "m_ID" and a value n that is the number of similar movies (i.e. movie ids) that we need to output from the function. This function will then return the n most similar movies to the movie 'm_ID'

In [18]:
```
def Jaccard(s1,s2):
    numerator = len(s1.intersection(s2))
```

```
    denominator = len(s1.union(s2))
    return numerator / denominator
```

In [19]:
```
def MostSimilar(m_ID,n):
    similarities = []
    users = UsersPerMovie[m_ID]
    for i in UsersPerMovie:
        if i == m_ID: continue
        sim = Jaccard(users,UsersPerMovie[i])
        # this is the similarity between number of users who watched movie m_ID and
        similarities.append((sim,i))
    similarities.sort(reverse=True)
    return similarities[:n]
```

In [20]:
```
UsersPerMovie[3] # number of users who watched movie 3
```

Out[20]:
```
{1687564,
 2334738,
 704538,
 294943,
 1130533,
 2015275,
 2646060,
 2170930,
 2531380,
 966716,
 983123,
 2588755,
 2154579,
 1056859,
 2646115,
 1204327,
 458859,
 966771,
 1269875,
 1081461,
 540794,
 32902,
 2318471,
 540807,
 1859725,
 368786,
 565401,
 303270,
 532649,
 2113709,
 2564278,
 1925306,
 2375867,
 1876156,
 745661,
 2359486,
 2572475,
 1007809,
 2490563,
 745688,
 2138332,
 1949919,
 262367,
 745706,
 1269998,
 1704175,
 1630448,
 966899,
 2253048,
 868600,
 237824,
 2105602,
 565510,
 786695,
 2269450,
 254222,
 401681,
 1646880,
 57633,
 2523426,
 2253090,
 1392933,
 835881,
 2031917,
```

```
2040110,
360757,
2384185,
352571,
1859908,
1130826,
2072907,
524628,
311641,
868700,
1868129,
2589029,
950631,
606570,
2597239,
794999,
2425210,
344444,
1270144,
762240,
1171843,
1507727,
2613654,
1565082,
41371,
2326945,
1450405,
1573289,
1589677,
156078,
1196471,
2630072,
1114552,
1278394,
1204665,
2539966,
2212303,
1040848,
25049,
2384355,
2531820,
213486,
1614320,
2040306,
1982967,
2171387,
2384389,
983558,
2621962,
1532433,
1163795,
2007573,
442911,
2032162,
573995,
639533,
2130480,
238130,
1401399,
115267,
2122313,
2433609,
2490971,
1876574,
```

```
1966687,
1204833,
328293,
1319527,
1622637,
819822,
1057394,
2089599,
1852040,
2572942,
475797,
2441882,
1122977,
1327792,
271028,
2622138,
2220732,
2081473,
74441,
565962,
393945,
1663717,
557801,
1057518,
1213178,
2237185,
1204995,
836357,
1835786,
1737484,
1221390,
2253583,
1508115,
803605,
1147677,
680740,
115498,
2253616,
1999678,
1819462,
1082193,
877394,
1024854,
713559,
1770331,
1188701,
1753949,
2261864,
2425709,
66414,
1606520,
2163577,
230265,
2245500,
533381,
2458504,
1344394,
1524620,
2614156,
1917839,
1663888,
2335640,
2573212,
1098655,
```

```
2294689,
1967013,
254887,
1393576,
222128,
762808,
787391,
402377,
336842,
50123,
1500112,
2180053,
1254363,
1082332,
1598429,
328674,
164845,
1328118,
795638,
410618,
2032635,
140284,
959486,
2491399,
1664010,
689166,
656399,
2483219,
1434651,
2180134,
975911,
1426472,
2229289,
1360936,
2130988,
263213,
1614895,
689205,
738379,
1672270,
2573391,
164942,
2090063,
435285,
722006,
1614939,
2450534,
91245,
1901682,
377975,
2638972,
664705,
500866,
541826,
820357,
631945,
1639566,
255120,
238740,
214166,
1262754,
885932,
1197233,
312506,
```

```
976059,
2467008,
1868998,
2352327,
2147527,
1598669,
672980,
1139925,
2139350,
517335,
1033433,
2393306,
1991906,
1541347,
1361124,
1107183,
214262,
1139969,
1459464,
673040,
2368790,
66852,
2311464,
312621,
828724,
1333,
1697078,
2630979,
2491723,
83278,
1115476,
582996,
902495,
779616,
763240,
1213801,
198000,
1639792,
574843,
2000256,
2336129,
370052,
853383,
820624,
1377693,
2213289,
1279413,
501174,
1787324,
320972,
386510,
1369550,
493009,
746965,
2295256,
1590745,
1689050,
2213342,
1967585,
1115632,
1189371,
1533440,
173571,
1394194,
```

```
2385435,
599584,
2344483,
1746477,
1386030,
771637,
1599030,
1099321,
1926723,
1312336,
132691,
1205848,
1304156,
1926750,
2303590,
755319,
2205307,
1607300,
1943182,
2213523,
665242,
517794,
1623720,
2213550,
222897,
100029,
2000582,
1361614,
2606799,
2197203,
1304280,
788185,
1025764,
1140453,
640743,
67315,
1255155,
2615035,
2426624,
2115337,
345869,
1959707,
2623268,
2090790,
1582905,
1984315,
2131774,
2377537,
313158,
2385738,
411471,
1566551,
395098,
2074458,
952156,
1689439,
788321,
1984358,
1533807,
1886074,
2393981,
1230729,
1607574,
2254751,
```

```
2590630,
2148273,
2615224,
223160,
2312124,
1828803,
2164676,
1968068,
165831,
1435594,
288727,
206809,
1386457,
780253,
1673185,
2303969,
1509354,
2631658,
2516973,
321519,
2598895,
1001461,
870391,
886790,
2222089,
133129,
10251,
1468428,
206865,
1353746,
870421,
1517590,
1550359,
2000928,
354344,
968745,
1763372,
968759,
1599561,
1255507,
1869907,
2369629,
1542242,
2140261,
223334,
1640549,
2025577,
2443370,
1460343,
1763451,
247940,
2164873,
1730699,
1534107,
2312349,
2017438,
1927329,
2386085,
2549926,
657582,
747695,
1214640,
305344,
2033859,
```

```
133322,
714960,
1370323,
1722588,
1444062,
575714,
477422,
1075444,
919805,
1100037,
2361606,
936199,
755976,
616720,
2378011,
2541854,
1648933,
788774,
1427751,
608554,
477486,
1870129,
2296114,
1698103,
223543,
2009400,
2369855,
231745,
2607433,
1403217,
698713,
387418,
1337699,
2124138,
215406,
2353520,
1763703,
2296192,
502145,
395655,
1321352,
1100170,
780687,
354704,
1370511,
1264021,
592281,
1182116,
2001318,
1935793,
453043,
27061,
1419705,
1788346,
772538,
1632700,
756162,
707012,
2468303,
747986,
1124822,
821727,
2583029,
2509302,
```

```
1190391,
322064,
789014,
1010200,
1419805,
461344,
109089,
2427438,
1477176,
2492984,
690746,
1272379,
2452028,
109124,
2607695,
1591925,
1215094,
1157750,
256633,
289419,
273047,
2370204,
1460897,
1739440,
1616563,
1591987,
1198785,
404170,
1796812,
1927891,
1903324,
1993442,
387819,
2427632,
1477365,
969462,
641785,
510719,
2534154,
2632461,
1674012,
1878820,
1444657,
371525,
1583947,
2083665,
1379159,
2567000,
1608536,
1657693,
1854303,
1559400,
396140,
2141037,
2370423,
2173816,
1035139,
2460555,
2616210,
207781,
854949,
1788839,
895914,
2444204,
```

```
183215,
2550711,
1149883,
2091987,
682963,
1772511,
478176,
1649641,
289781,
371701,
2550775,
1797112,
1289218,
2255880,
723983,
699412,
1240085,
1813526,
789530,
969759,
1641505,
2575399,
1444903,
494633,
2526250,
871489,
1395781,
2419782,
1420370,
199769,
1412189,
248932,
2083947,
216174,
1830000,
683123,
117878,
871548,
1985660,
1387650,
126083,
134288,
871580,
1871017,
2460844,
1985710,
101554,
1698999,
1461435,
1854652,
191678,
1707198,
797889,
1248452,
1854660,
1166536,
1551571,
1871063,
2436327,
273641,
1092852,
1248503,
1174779,
1404157,
```

```
2510089,
2043149,
707854,
1150223,
593167,
478484,
544022,
1404191,
2190625,
1477923,
675111,
1150258,
52540,
1682749,
2485566,
1830211,
1731912,
2583888,
1215825,
1215832,
1355097,
1469792,
2264426,
1830265,
1969536,
1076614,
1928584,
1551753,
306569,
2272652,
2567571,
1887644,
2354593,
1011106,
552356,
2387369,
1756597,
306622,
937412,
781779,
2584024,
2616793,
576990,
478688,
1404389,
699878,
257517,
200177,
1756658,
1797618,
1969676,
994829,
1404431,
1142291,
1822229,
1494569,
60973,
593454,
2494005,
1764923,
323134,
470596,
2133574,
798281,
```

```
323148,
650828,
2043472,
798296,
2616920,
822886,
1977959,
364140,
749164,
1961586,
896636,
1650301,
1945220,
429701,
421509,
3718,
388752,
2190996,
1822357,
1240730,
1543836,
331430,
1207990,
1961675,
560851,
1543897,
945888,
683751,
634604,
1478381,
44783,
2191089,
1838839,
380673,
2641668,
569099,
601868,
1560333,
454417,
2010898,
1068821,
667422,
962342,
749353,
1486636,
2625327,
372528,
618288,
2117432,
2535238,
1052493,
1691501,
2158448,
1257336,
2625405,
1527679,
1658752,
1077120,
831364,
44937,
1191821,
1822608,
1650580,
1994646,
```

```
511899,
282525,
1544094,
2600868,
118694,
1634219,
356269,
126899,
2625471,
1912781,
184273,
1331154,
872408,
1470425,
1880039,
1208313,
1765372,
2551806,
528384,
1740800,
2002948,
2297863,
577547,
167949,
561184,
1241131,
1085488,
1314869,
2338873,
1462330,
659526,
1372231,
1560653,
2117724,
2429036,
282736,
307313,
2592894,
290951,
913558,
1511584,
1863843,
356515,
2396324,
372910,
2322625,
1437916,
1134816,
307427,
782566,
618726,
1945838,
545026,
1511683,
2068749,
1929487,
594196,
1904916,
1560852,
1716507,
1618217,
1265961,
340279,
1249592,
```

```
1929531,
28995,
61765,
700747,
1716556,
2601294,
1036626,
1233239,
2412908,
2085230,
1798520,
790920,
2363785,
962955,
323983,
635285,
1708442,
1053090,
381346,
1331628,
1241521,
53694,
963009,
2191815,
102859,
2175436,
979424,
520675,
504296,
1167852,
2118126,
766450,
791029,
1610234,
807418,
2306557,
2552319,
2134527,
946691,
2118159,
864786,
1036823,
1888794,
414236,
479779,
1921572,
217658,
1733188,
78404,
1176132,
1765966,
815695,
1176145,
2052691,
512598,
2470497,
2003554,
1528420,
1774181,
2028133,
512629,
2609783,
201338,
2232958,
```

```
2601604,
340618,
1806991,
234141,
553632,
373416,
823979,
4783,
586421,
373441,
406219,
2241241,
709342,
439011,
1045221,
250597,
2142951,
1594095,
1389295,
316155,
1364733,
1086213,
2437901,
504593,
2331421,
1733406,
2519847,
2224936,
504624,
2331444,
2470712,
2118461,
299857,
2446163,
78684,
2126686,
545639,
979820,
1012590,
86901,
1446775,
2282361,
152441,
2495357,
2192258,
2216837,
1799047,
1749903,
1241999,
2012052,
2274199,
766872,
2331547,
586652,
127911,
1897386,
693162,
1717162,
1168312,
2200505,
1037245,
1373119,
1684416,
553931,
```

```
86987,
480206,
2307036,
2569181,
136160,
660454,
168939,
2159596,
1561588,
586741,
2135038,
332805,
46086,
496645,
2552842,
2421781,
513050,
963611,
1872926,
668703,
1045551,
37939,
1250364,
701514,
2331728,
78931,
308307,
2159701,
2249828,
1627242,
1217660,
283774,
357507,
1307784,
152713,
1741962,
603277,
46222,
971925,
570522,
...}
```

In [21]:

```
# the following gives the number of movies in the dictionary UsersPerMovie.
# against each movie i, we have a list of users who watched and rated that
# movie
for i in UsersPerMovie:
    print(i)
```

```
3
8
16
17
18
26
28
30
32
33
44
45
46
47
48
52
55
56
57
58
68
76
77
78
79
83
84
97
104
108
110
111
118
122
127
133
138
143
148
152
156
165
166
167
171
173
175
178
180
181
185
187
188
189
191
197
199
201
208
209
213
215
216
223
```

```
225
232
238
239
240
241
242
248
252
253
255
256
257
262
268
269
270
273
275
281
283
285
289
290
295
297
299
304
305
311
312
313
316
329
330
331
334
337
341
344
345
348
353
356
357
358
359
360
361
362
367
371
375
378
381
385
390
393
395
398
400
401
405
406
```

```
408
413
416
418
420
422
424
425
426
427
431
432
433
436
438
442
443
445
452
454
456
457
459
463
468
471
473
474
476
477
482
483
486
488
489
494
499
501
504
505
516
517
518
524
528
533
534
535
548
550
551
554
561
563
564
569
571
575
577
579
580
582
585
586
```

```
588
599
600
607
615
621
629
631
636
638
643
645
646
652
657
658
659
660
661
662
668
670
672
674
676
677
680
681
682
686
689
692
693
696
700
705
707
708
711
717
720
722
723
725
730
731
732
733
734
746
748
749
750
751
752
754
758
759
760
761
762
763
774
788
```

```
789
798
807
808
809
811
817
818
819
822
825
829
831
832
833
840
843
846
850
851
859
862
872
883
886
889
894
895
896
897
900
907
908
918
919
931
937
940
942
952
953
954
956
961
962
963
964
971
977
978
985
986
988
989
990
992
993
994
996
1001
1011
1012
1020
1021
```

```
1022
1026
1027
1032
1035
1043
1044
1045
1046
1047
1050
1055
1058
1060
1066
1068
1073
1074
1075
1080
1087
1089
1092
1094
1096
1098
1100
1102
1103
1104
1110
1111
1116
1120
1123
1126
1129
1130
1132
1134
1138
1140
1142
1144
1145
1148
1151
1160
1163
1172
1173
1174
1175
1176
1179
1180
1193
1200
1201
1202
1208
1210
1216
1218
```

```
1220
1221
1222
1224
1226
1234
1236
1238
1245
1250
1252
1255
1256
1260
1262
1264
1266
1267
1270
1274
1277
1282
1289
1291
1292
1293
1295
1297
1298
1299
1300
1305
1307
1314
1324
1329
1336
1357
1359
1363
1364
1365
1367
1370
1373
1375
1384
1392
1394
1395
1399
1401
1406
1408
1409
1412
1421
1425
1428
1432
1435
1441
1450
1455
```

```
1466
1467
1470
1471
1476
1479
1481
1482
1495
1502
1503
1507
1509
1513
1514
1518
1521
1525
1528
1530
1532
1542
1543
1547
1552
1553
1558
1561
1571
1578
1582
1585
1589
1590
1594
1595
1599
1600
1602
1604
1607
1610
1615
1618
1625
1627
1632
1633
1637
1638
1642
1645
1646
1650
1652
1655
1656
1659
1660
1661
1664
1665
1666
1668
```

```
1673
1682
1689
1692
1693
1694
1700
1703
1704
1707
1708
1709
1719
1721
1722
1728
1734
1735
1741
1743
1744
1746
1754
1756
1757
1759
1760
1765
1766
1770
1771
1780
1783
1788
1790
1792
1794
1795
1798
1799
1800
1803
1804
1810
1814
1816
1819
1821
1832
1833
1834
1837
1839
1843
1844
1845
1848
1851
1853
1854
1856
1860
1861
1862
```

```
1865
1866
1867
1877
1878
1882
1884
1885
1890
1901
1902
1905
1912
1913
1918
1923
1925
1931
1939
1947
1955
1956
1959
1962
1963
1969
1971
1972
1974
1975
1976
1983
1994
1998
2000
2001
2009
2012
2015
2016
2019
2033
2037
2040
2043
2045
2047
2051
2057
2072
2077
2078
2080
2095
2102
2103
2104
2109
2111
2112
2113
2114
2122
2128
```

```
2129
2130
2132
2135
2136
2137
2139
2140
2144
2149
2152
2153
2161
2162
2163
2164
2167
2171
2172
2173
2174
2176
2177
2178
2181
2182
2186
2188
2189
2192
2195
2199
2200
2209
2212
2216
2217
2218
2228
2235
2251
2252
2254
2256
2262
2266
2269
2276
2284
2285
2287
2290
2300
2302
2304
2310
2314
2316
2319
2326
2329
2331
2332
2335
```

```
2337
2338
2339
2340
2342
2346
2348
2350
2352
2356
2360
2362
2366
2370
2371
2372
2375
2376
2379
2380
2383
2385
2386
2388
2389
2391
2394
2395
2400
2405
2408
2409
2411
2418
2422
2423
2430
2435
2440
2441
2443
2448
2452
2456
2457
2460
2462
2464
2465
2467
2470
2471
2475
2478
2482
2493
2495
2498
2499
2505
2512
2518
2519
2521
```

```
2523
2525
2527
2528
2532
2533
2539
2548
2551
2554
2558
2559
2566
2568
2572
2574
2577
2578
2580
2585
2590
2594
2595
2599
2601
2603
2612
2613
2617
2622
2638
2640
2643
2651
2654
2659
2660
2662
2668
2674
2675
2680
2690
2698
2699
2701
2705
2708
2711
2712
2724
2727
2731
2732
2734
2735
2738
2743
2749
2751
2755
2757
2762
2769
```

```
2771
2775
2778
2779
2780
2782
2783
2791
2795
2796
2798
2800
2803
2804
2809
2813
2823
2828
2831
2834
2847
2848
2851
2852
2861
2862
2865
2866
2868
2870
2872
2873
2874
2876
2880
2882
2889
2890
2897
2899
2900
2905
2908
2912
2913
2916
2920
2921
2922
2927
2938
2940
2942
2944
2945
2949
2953
2954
2955
2960
2965
2967
2972
2974
```

```
2980
2981
2986
2988
2989
2992
3003
3005
3009
3013
3015
3017
3021
3023
3030
3035
3043
3046
3047
3054
3058
3071
3077
3078
3079
3082
3084
3085
3086
3090
3093
3095
3098
3102
3106
3113
3120
3124
3128
3135
3138
3147
3148
3150
3151
3153
3159
3160
3161
3165
3168
3170
3174
3181
3182
3183
3190
3192
3197
3198
3207
3208
3210
3212
```

```
3216
3220
3222
3223
3224
3225
3226
3229
3232
3237
3239
3242
3253
3254
3256
3264
3265
3266
3267
3268
3269
3274
3275
3276
3282
3285
3290
3295
3301
3309
3310
3312
3313
3314
3315
3320
3321
3324
3326
3331
3333
3342
3347
3348
3350
3355
3364
3368
3369
3371
3376
3379
3385
3391
3394
3397
3398
3403
3404
3405
3408
3410
3414
3416
```

```
3418
3420
3422
3423
3427
3433
3434
3437
3439
3441
3444
3446
3451
3453
3456
3463
3466
3469
3475
3478
3480
3481
3487
3489
3491
3495
3496
3500
3505
3506
3509
3510
3513
3515
3521
3522
3523
3525
3526
3529
3532
3535
3538
3541
3542
3544
3551
3557
3559
3563
3565
3567
3570
3571
3573
3579
3581
3582
3583
3587
3590
3593
3595
3599
```

```
3605
3608
3610
3611
3612
3615
3617
3623
3624
3626
3638
3640
3642
3648
3649
3650
3657
3662
3668
3670
3672
3680
3684
3685
3686
3689
3696
3701
3703
3710
3713
3714
3715
3716
3725
3728
3730
3733
3736
3740
3742
3743
3756
3758
3763
3765
3766
3768
3769
3773
3775
3777
3782
3787
3796
3798
3801
3807
3810
3812
3814
3816
3817
3824
```

```
3825
3826
3828
3830
3833
3835
3840
3841
3854
3858
3860
3862
3863
3864
3870
3875
3879
3880
3886
3887
3888
3890
3893
3894
3900
3903
3905
3907
3917
3920
3921
3923
3925
3926
3927
3928
3934
3935
3936
3938
3942
3946
3949
3954
3955
3958
3960
3962
3966
3970
3972
3975
3984
3986
3988
3990
3999
4007
4011
4012
4027
4031
4033
4040
```

```
4042
4043
4049
4054
4055
4056
4060
4064
4067
4069
4072
4078
4080
4086
4089
4092
4098
4100
4106
4109
4115
4123
4127
4131
4135
4136
4138
4141
4144
4145
4147
4149
4155
4157
4159
4161
4166
4168
4171
4173
4177
4179
4185
4197
4201
4207
4210
4214
4216
4219
4225
4227
4237
4238
4247
4248
4249
4253
4255
4256
4260
4262
4266
4267
```

```
4269
4271
4281
4282
4284
4287
4289
4290
4298
4299
4302
4306
4310
4315
4330
4331
4339
4341
4345
4346
4352
4353
4354
4355
4356
4360
4364
4366
4369
4372
4374
4377
4380
4383
4384
4386
4387
4389
4390
4392
4393
4396
4402
4405
4407
4411
4413
4418
4420
4427
4429
4432
4438
4441
4442
4450
4454
4460
4463
4465
4472
4474
4478
4479
```

```
4485
4488
4490
4492
4493
4496
```

Getting a recommendation:

In [22]:
```
m_ID = 3
MostSimilar(m_ID,5)
```

Out[22]:
```
[(0.12352209085252022, 3301),
 (0.11056751467710371, 4450),
 (0.08810335546384353, 1479),
 (0.08634175691937425, 3404),
 (0.08425055033019811, 4155)]
```

In [23]:
```
print(i)
m_ID
```

```
4496
```

Out[23]:
```
3
```

In [24]:
```
# Now, we need to find the movie names using the the dataset: movie_data

# MovieNames[movie_data[movie]] = d['Name']
i=1
MovieNames = [movie_data[i[1]-1] for i in MostSimilar(m_ID,5)]
```

In [25]:
```
#MostSimilar(m_ID,5)[0][1]
#MostSimilar(m_ID,5)[0]
MostSimilar(m_ID,5)
```

Out[25]:
```
[(0.12352209085252022, 3301),
 (0.11056751467710371, 4450),
 (0.08810335546384353, 1479),
 (0.08634175691937425, 3404),
 (0.08425055033019811, 4155)]
```

In [26]:
```
MovieNames
```

Out[26]:
```
[{'Movie_ID': '3301', 'Year': '1994', 'Name': 'Burnt by the Sun'},
 {'Movie_ID': '4450', 'Year': '1987', 'Name': 'Pelle the Conqueror'},
 {'Movie_ID': '1479', 'Year': '2002', 'Name': 'Man on the Train'},
 {'Movie_ID': '3404', 'Year': '2000', 'Name': 'Not One Less'},
 {'Movie_ID': '4155', 'Year': '2000', 'Name': 'East/West'}]
```

Part 3: Collaborative - Filtering Based Rating Estimation

This part is to make predictions about user's ratings. Specifically, a user's rating for a movie is assumed to be a weighted sum of their previous ratings, weighted by how similar the query movie is to each of their previous views.

In [27]:
```
ratingsPerUser = defaultdict(list)
ratingsPerMovie = defaultdict(list)

for d in dataset:
    user,movie = d['User_ID'], d['Movie_ID']
    ratingsPerUser[user].append(d)
    ratingsPerMovie[movie].append(d)
```

In [28]:
```
# Calculate the mean rating of the entire dataset
import numpy as np
mean_rating = np.mean([dataset[i]['Rating'] for i in range(len(dataset))])
mean_rating
```

Out[28]:
```
3.590569909383486
```

In [29]:
```
dataset[15]
```

Out[29]:
```
{'User_ID': 1694958, 'Rating': 3, 'Movie_ID': 3}
```

In [30]:
```
# Function to predict rating of a movie based on a user and a movie

def PredictRating(user,movie):
    ratings = []
    similarities = []
    for d in ratingsPerUser[user]: # i.e. all the movies and ratings by the user.
        i = d['Movie_ID']
        if i == movie: continue
        ratings.append(d['Rating'])
        users = UsersPerMovie[movie]
        similarities.append(Jaccard(users,UsersPerMovie[i]))
    if(sum(similarities) > 0):
        weightedRatings = [(x*y) for x,y in zip(ratings, similarities)]
        return sum(weightedRatings) / sum(similarities)
    else:
        return mean_rating
```

In [31]:
```
user, movie = dataset[10]['User_ID'], dataset[10]['Movie_ID']
PredictRating(user,movie)
```

Out[31]:
```
3.553601713242338
```

In [32]:
```
dataset[0]
```

Out[32]:
```
{'User_ID': 712664, 'Rating': 5, 'Movie_ID': 3}
```

Part 4: Evaluating Performance By calculating MSE

In [33]:
```
def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions, labels)]
    return sum(differences) / len(differences)
```

In [34]:
```
p_mean = [mean_rating for d in dataset]
```

In [36]:
```
# To reduce computation time, the size of the dataset has been reduced to 100
predictions = []
count = 0
for d in dataset[:1000]:
    predictions.append(PredictRating(d['User_ID'], d['Movie_ID']))
    count= count+1
```

In [37]:
```
count
```

Out[37]:
```
1000
```

In [38]:
```
PredictRating(d['User_ID'], d['Movie_ID'])
```

Out[38]: 3.0793295588716907

In [39]:
```
# our predictions from using the function Predictrating above:
#predictions = [PredictRating(d['User_ID'], d['Movie_ID']) for d in dataset]
```

In [40]:
```
labels = [d['Rating'] for d in dataset[:1000]]
```

In [41]:
```
MSE(p_mean, labels)
```

Out[41]: 0.9531906895087604

In [42]:
```
MSE(predictions, labels)
```

Out[42]: 0.718229871418949

In this case, our rating prediction model was better in terms of MSE than predicting the mean rating.

Latent Factor based Recommender Systems

In [43]:
```
# This is from above:

ratingsPerUser = defaultdict(list)
ratingsPerMovie = defaultdict(list)

for d in dataset:
    user,movie = d['User_ID'], d['Movie_ID']
    ratingsPerUser[user].append(d)
    ratingsPerMovie[movie].append(d)
```

In [44]:
```
# Get the length of the dataset and the above dictionaries
# due to computational issue, the size of the dataset was reduced
dataset = dataset[:1000]

N = len(dataset)
nUsers = len(ratingsPerUser) # number of users
nMovies = len(ratingsPerMovie) # number of movies

# Get the list of keys
users = list(ratingsPerUser.keys())
movies = list(ratingsPerMovie.keys())

# calculate the variables of our model: alpha, userBiases and MovieBiases
alpha = sum([d['Rating'] for d in dataset])/ len(dataset)
UserBiases = defaultdict(float)
MovieBiases = defaultdict(float)

def MSE(predictions,labels):
    differences = [(x-y)**2 for x,y in zip(predictions, labels)]
    return sum(differences)/len(differences)
```

In [45]:
```
# LFM = Latent Factor Model
def Prediction_LFM(user,movie):
    return alpha + UserBiases[user] + MovieBiases[movie]
```

In [46]:
```
# unpack function to unpack the vector theta into the offset and bias terms

def unpack(theta):
    global alpha
```

```
    global UserBiases
    global MovieBiases
    alpha = theta[0]
    UserBiases = dict(zip(users, theta[1:nUsers+1])) # is the +1 reqd here?
    MovieBiases = dict(zip(movies, theta[1+nUsers:]))
```

In [47]:
```
# we will find theta such that the foll cost function is optimised:
# this gives the regularized MSE of the solution:

def cost(theta, labels, lamb):
    unpack(theta)
    predictions = [Prediction_LFM(d['User_ID'], d['Movie_ID']) for d in dataset]
    cost = MSE(predictions, labels)
    print("MSE = " + str(cost))
    for u in UserBiases:
        cost += lamb*UserBiases[u]**2
    for i in MovieBiases:
        cost += lamb*MovieBiases[i]**2
    return cost
```

In [48]:
```
# the derivative function:
def derivative(theta, labels, lamb):
    unpack(theta)
    N = len(dataset)
    dalpha = 0
    dUserBiases = defaultdict(float)
    dMovieBiases = defaultdict(float)
    for d in dataset:
        u,i = d['User_ID'], d['Movie_ID']
        pred = Prediction_LFM(u,i)
        diff = pred - d['Rating']
        dalpha += 2/N*diff
        dUserBiases[u] += 2/N*diff
        dMovieBiases[i] += 2/N*diff
    for u in UserBiases:
        dUserBiases[u] += 2*lamb*UserBiases[u]
    for i in MovieBiases:
        dMovieBiases[i] += 2*lamb*MovieBiases[i]
    dtheta = [alpha] + [dUserBiases[u] for u in users] + [dMovieBiases[i] for i in
    return numpy.array(dtheta)
```

In [49]:
```
# MSE when always predicting mean
p_mean = [alpha for d in dataset]
labels = [d['Rating'] for d in dataset]

MSE(p_mean, labels)
```

Out[49]: 0.9525439999999995

In [50]:
```
# The gradient descent model using scipy:
scipy.optimize.fmin_l_bfgs_b(cost, [alpha] + [0.0]*(nUsers+nMovies),
                             derivative, args = (labels, 0.001))
```

```
MSE = 0.9525439999999995
MSE = 1.951199431486239
MSE = 0.9805295329824282
MSE = 0.9536613460427796
MSE = 0.9525871148504756
MSE = 0.9525446716409175
MSE = 0.9525437663728146
MSE = 0.9525439339219235
MSE = 0.9525439853029469
MSE = 0.952543996863862
MSE = 0.952543999336417
MSE = 0.9525439998598355
MSE = 0.9525439999704095
MSE = 0.9525439999937427
MSE = 0.952543999998686
MSE = 0.952543999999725
MSE = 0.9525439999999505
MSE = 0.9525439999999831
MSE = 0.9525440000000086
MSE = 0.9525439999999844
MSE = 0.9525439999999936
```

Out[50]:
```
(array([3.616, 0.   , 0.   , ..., 0.   , 0.   , 0.   ]),
 0.9525439999999936,
 {'grad': array([ 3.616e+00, -2.768e-03, -7.680e-04, ...,  0.000e+00,  0.000e+00,
          0.000e+00]),
  'task': 'ABNORMAL_TERMINATION_IN_LNSRCH',
  'funcalls': 21,
  'nit': 0,
  'warnflag': 2})
```

The Complete Latent Factor Model:

In [51]:
```python
# For each user and item, we now have an additional low-dimensional
# descriptor of user's preferences of dimension K = gamma_u. And, a
# K dimensional representation of each item = gamma_k
# Model: f(u,i) = alpha + beta_u + beta_i + gamma_u*gamma_i
# the gamma term is the interaction term which tries to answer :
# Are the user preferences consistent with the item's properties?

UserBiases = defaultdict(float)
MovieBiases = defaultdict(float)
UserGamma = {}
MovieGamma = {}

K = 2
```

In [52]:
```python
for u in ratingsPerUser:
    UserGamma[u] = [random.random()*0.1 - 0.05 for k in range(K)]

for i in ratingsPerMovie:
    MovieGamma[i] = [random.random()*0.1 - 0.05 for k in range(K) ]
```

In [53]:
```python
# unpack function taking into account additional terms

def unpack(theta):
    global alpha
    global UserBiases
    global MovieBiases
    global UserGamma
    global MovieGamma
    index = 0
    alpha = theta[index]
    index += 1
```

```
    UserBiases = dict(zip(users, theta[index: index+nUsers]))
    index += nUsers
    MovieBiases = dict(zip(movies,theta[index:index+nMovies]))
    index += nMovies
    for u in users:
        UserGamma[u] = theta[index:index+K]
        index+=K
    for m in movies:
        MovieGamma[m] = theta[index:index+K]
        index += K
```

In [54]:
```
def inner(x,y):
    return sum([a*b for a,b in zip(x,y)])
```

In [55]:
```
def prediction(user,movie):
    return alpha + UserBiases[user] + MovieBiases[movie]  + inner(UserGamma[user],

def cost(theta,labels, lamb):
    unpack(theta)
    predictions = [prediction(d['User_ID'], d['Movie_ID'])]
    cost = MSE(predictions, labels)
    print("MSE=" + str(cost))
    for u in users:
        cost+=lamb*UserBiases[u]**2
        for k in range(K):
            cost += lamb*UserGamma[u][k]**2
    for i in movies:
        cost += lamb* MovieBiases[i] **2
        for k in range(K):
            cost += lamb*MovieGamma[i][k]**2
    return cost
```

In [56]:
```
def derivative(theta, labels, lamb):
    unpack(theta)
    N = len(dataset)
    dalpha = 0
    dUserBiases = defaultdict(float)
    dMovieBiases = defaultdict(float)
    dUserGamma = {}
    dMovieGamma = {}
    for u in ratingsPerUser:
        dUserGamma[u] = [0.0 for k in range(K)]
    for i in ratingsPerMovie:
        dMovieGamma[i] = [0.0 for k in range(K)]
    for d in dataset:
        u,i = d['User_ID'], d['Movie_ID']
        pred = prediction(u, i)
        diff = pred - d['Rating']
        dalpha += 2/N*diff
        dUserBiases[u] += 2/N*diff
        dMovieBiases[i] += 2/N*diff
        for k in range(K):
            dUserGamma[u][k] += 2/N*MovieGamma[i][k]*diff
            dMovieGamma[i][k] += 2/N*UserGamma[u][k]*diff
    for u in UserBiases:
        dUserBiases[u] += 2*lamb*UserBiases[u]
        for k in range(K):
            dUserGamma[u][k] += 2*lamb*UserGamma[u][k]
    for i in MovieBiases:
        dMovieBiases[i] += 2*lamb*MovieBiases[i]
        for k in range(K):
            dMovieGamma[i][k] += 2*lamb*MovieGamma[i][k]
    dtheta = [dalpha] + [dUserBiases[u] for u in users] + [dMovieBiases[i] for i ir
```

```
    for u in users:
        dtheta += dUserGamma[u]
    for i in movies:
        dtheta += dMovieGamma[i]
    return numpy.array(dtheta)
```

In [57]:
```
MSE(p_mean, labels)
```

Out[57]:
```
0.9525439999999995
```

In [58]:
```
scipy.optimize.fmin_l_bfgs_b(cost, [alpha] +
                                  [0.0]*(nUsers+nMovies) +
                                  [random.random() * 0.1 - 0.05 for k in range(K*(
                             derivative, args = (labels, 0.001), maxfun = 10, maxit
```

```
MSE=1.9148411401272962
MSE=1.9135443686957903
MSE=1.908351404372158
MSE=49.902015833867964
MSE=7.554367849469394
MSE=2.0606398692270873
MSE=1.9104139175357417
MSE=1.9083790176369813
MSE=1.9083517740075149
MSE=1.908351409320141
MSE=1.9083514044383916
MSE=1.908351404373045
MSE=1.90835140437217
MSE=1.908351404372158
MSE=1.9083514043721652
MSE=1.908351404372158
MSE=1.908351404372158
MSE=1.9083514043721617
MSE=1.908351404372158
MSE=1.908351404372158
MSE=1.9083514043721603
MSE=1.908351404372158
MSE=1.908351404372158
MSE=1.9346718047939926
MSE=1.908799954878752
MSE=1.9083590345464339
MSE=1.9083515341630124
MSE=1.908351406579925
MSE=1.9083514044097125
MSE=1.9083514043727972
MSE=1.908351404372169
MSE=1.908351404372158
MSE=1.9083514043721652
MSE=1.908351404372158
MSE=1.9083514043721628
MSE=1.908351404372158
MSE=1.908351404372158
MSE=1.9083514043721603
MSE=1.908351404372158
MSE=1.908351404372158
MSE=1.9083514043721592
MSE=1.908351404372158
MSE=1.908351404372158
```

Out[58]:
```
(array([ 3.61840654e+00,  2.00307767e-01,  5.55273909e-02, ...,
        -4.04355908e-02, -1.45345873e-02,  2.37388126e-04]),
 2.105026656699331,
 {'grad': array([ 9.49322611e-03, -1.95815467e-03, -5.37643812e-04, ...,
         -8.08711815e-05, -2.90691747e-05,  4.74776251e-07]),
  'task': 'ABNORMAL_TERMINATION_IN_LNSRCH',
  'funcalls': 43,
  'nit': 1,
  'warnflag': 1})
```

Using the latent factor method, the rating prediction model is worse than predicting ratings using the dataset mean rating in terms of the MSE.

In [ ]:

In [ ]:

In [ ]: