Project 2

IMDb Database Exploration

Professor - Vwani Roychowdhury

Class - EE232E Graphs and Network Flows

June 12, 2015

Yulia Sunyoto | 504042206

Seyoon Park | 304422269

1. Download actors.list.gz, actresses.list.gz (or use the actor_movies.txt and actress_movies.txt files from the cleaned up data), merge those 2 lists into one file, and remove all actors/actressess with less than 5 (so actors who have acted in four or fewer number of movies) movies; Note that you will have to parse the data in these lists as accurately as possible to extract the entities consistently and create the network. So plan onspending some time in cleaning the data set.

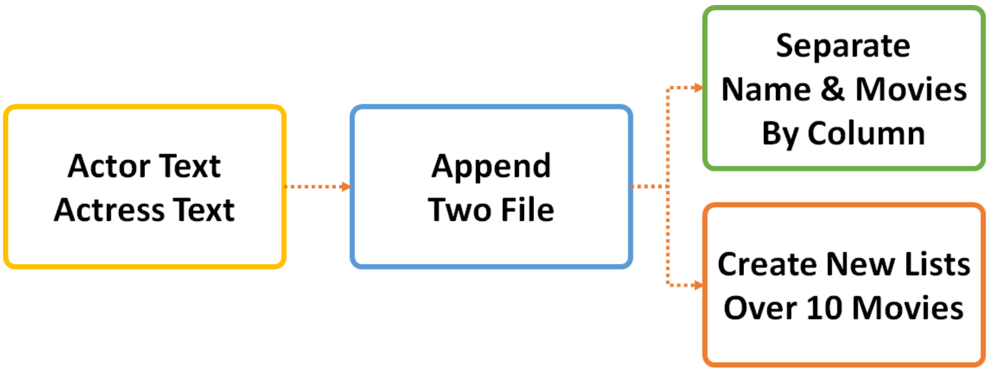
[Code]

```
actor_list <- list()
actress_list <- list()

conn_actor=file("C:/down/project_2_data/actor_movies.txt",open="r")
actor_list <- strsplit(readLines(conn_actor),"\t\t")
close(conn_actor)

conn_actress=file("C:/down/project_2_data/actress_movies.txt",open="r")
actress_list <- strsplit(readLines(conn_actress),"\t\t")
close(conn_actress)

total_list <- append(actor_list,actress_list)

start <- 1
percent <- 0
if (length(name_list) == 0 && length(movie_list) == 0) {
   name_list <- list()
   movie_list <- list()
} else {
   name_length <- length(name_list)
   movie_length <- length(movie_list)
   if (movie_length == name_length) {
      start <- length(name_list)+1
   } else {
      name_list <- correction_func(name_list,1)
      movie_list <- correction_func(movie_list,2)
      name_length <- length(name_list)
      movie_length <- length(movie_list)
      start <- length(name_list)+1
   }

}
t_number <- length(total_list)
for (i in start:length(total_list)) {
   #name_list[[length(name_list)+1]] <- total_list[[i]][1]
   #movie_list[[length(movie_list)+1]] <- total_list[[i]][2:length(total_list[[i]])]

   name_list <- append(name_list,total_list[[i]][1])
   movie_list <- append(movie_list,list(total_list[[i]][2:length(total_list[[i]])]))

   if (trunc(i/t_number*10000) > percent) {
      percent <- trunc(i/t_number*10000)
      cat((percent/100),'%','\n')
   } else {

   }

}
```

```r
name_list[[length(name_list)]] <- NULL
movie_list[[length(movie_list)]] <- NULL

correction_func <- function(list,flag) {
    if (flag == 1) {
        # Correct for Name
        percent <- 0
        t_number <- length(total_list)
        for (i in 1:length(total_list)) {

            if (identical(total_list[[i]][2:length(total_list[[i]])],list[[i]]) == 'FALSE') {
                first_part_tmp <- list[1:(i-1)]
                second_part_tmp <- list[i:length(list)]
                first_part_tmp <- list[1:(i-1)]
                second_part_tmp <- list[i:length(list)]
                first_part_tmp <- append(first_part_tmp,total_list[[i]][1])
                first_part_tmp <- append(first_part_tmp,second_part_tmp)
                list <- first_part_tmp
                rm(first_part_tmp)
                cat(i,' Error Correct\n')
            }
            if (trunc(i/t_number*10000) > percent) {
                percent <- trunc(i/t_number*10000)
                cat((percent/100),'%','\n')
            } else {

            }
        }
    } else if (flag == 2) {
        # Correct for Movie
        percent <- 0
        t_number <- length(total_list)
        for (i in 1:length(total_list)) {

            if (identical(total_list[[i]][2:length(total_list[[i]])],list[[i]]) == 'FALSE') {
                first_part_tmp <- list[1:(i-1)]
                second_part_tmp <- list[i:length(list)]
                first_part_tmp <- list[1:(i-1)]
                second_part_tmp <- list[i:length(list)]
                first_part_tmp <- append(first_part_tmp,list(total_list[[i]][2:length(total_list[[i]])]))
                first_part_tmp <- append(first_part_tmp,second_part_tmp)
                list <- first_part_tmp
                rm(first_part_tmp)
                cat(i,' Error Correct\n')
            }
            if (trunc(i/t_number*10000) > percent) {
                percent <- trunc(i/t_number*10000)
                cat((percent/100),'%','\n')
            } else {

            }
        }
    }
}
```

```
# Erase NA in movie_list
movie_list2 <- movie_list
if (exists(i_NA)) {
   continue_NA <- i_NA
} else {
   continue_NA <- 1
}
for (i_NA in continue_NA:length(movie_list2)) {
   if (length(which(is.na(movie_list2[[i_NA]]))) == 1) {
      movie_list2[[i_NA]] <- movie_list2[[i_NA]][-which(is.na(movie_list2[[i_NA]]))]
   } else {

   }
}
cat("Complete")


# Filtering Movie List -> Extract Only Title (Some includes Role Name after year)
total_count <- length(movie_list)
percent <- 0
start_row <- 1
start_col <- 1
if (exists("m_row") && exists("m_col")) {
   start_row <- m_row
   start_col <- m_col
}
for (m_row in 1:length(movie_list)) {
   for (m_col in 1:length(movie_list[[m_row]])) {
      movie_list[[m_row]][m_col] <- unlist(strsplit(movie_list[[m_row]][m_col], ")"))[1]
   }

   if (trunc(m_row/total_count*10000) > percent) {
      percent <- trunc(m_row/total_count*10000)
      cat((percent/100),'%','\n')
   } else {

   }
}

# Extract Actor/Actress Have 10 Movies
total_list <- append(actor_list,actress_list)
extracted_name_list <- list()
for (i in 1:length(total_list)) {
   if (length(total_list[[i]]) >= 11) {
      extracted_name_list[[length(extracted_name_list)+1]] <- total_list[[i]][1]
      #extracted_movie_list[[length(extracted_movie_list)+1]] <- total_list[[i]][2:length(total_list[[i]])]
   }
}

extracted_movie_list <- vector("list", length(extracted_name_list))
percent <- 0
t_number <- length(extracted_name_list)
for (i in 1:length(extracted_name_list)) {
   extracted_movie_list[[i]] <- movie_list[[extracted_name_index_list[[i]]]]
```

```
    if (trunc(i/t_number*10000) > percent) {
      percent <- trunc(i/t_number*10000)
      cat((percent/100),'%','\n')
    }
}
```



For preparing the basic data, we need to merge two actor and actress file. And then we separate the name part and movies part similar to a column. The reason why we separate the name and movies are that there are many cases for searching by actor or actress name. If the actor or actress name is not grouped and combined with movies then it takes a long time to search. Moreover, we need to make an actor and actress index. Therefore, the entire name list is essential for later work.

And we need to extract some nodes who have more than 5 movies. However, we increased the threshold to 10 because the extracted data was too much. The number of extracted lists was 113124.

extracted_movie_list	Large list (113124 elements, 269.2 Mb)
extracted_name_list	Large list (113124 elements, 12.8 Mb)

This two lists will be used for creating edge list in Part 2.

```
(500) Days of Summer (2009)
(500) Days of Summer (2009)  (uncredited)
(500) Days of Summer (2009)  (voice)
'Fraid Cat (1914)
'Fraidy Cat (1951)  (uncredited)
'From Mad to Worse' (1957)  (voice) (uncredited)
'Fun on a Week-End' (1947)
'Fun on a Week-End' (1947)  (uncredited)
'G' Men (1935)
'G' Men (1935)  (uncredited)
'G.' (2014)
'Gatillo Veloz' en 'Los Malditos' (1966)
'Gatillo Veloz' en 'Los Malditos' (1966)  (as Hnos. Carrion)
```

While we make the basic dataset. We found a problem on the movie titles. We tried to make a unique movie index lists for movie indexing. We eliminated duplicates by ‘unique’ function. However, there were a lot of duplicates. In above Figure, you can see that there are an additional explain about the role of movie or it is credited or not. Those additional parts make the comparing elements in the list be very difficult. Therefore, we modified the movie list to get rid of the additional part after a year.

```
: chr "'95 (2013"
: chr "'A' for Effort (2013"
: chr "'A' gai wak (1983"
: chr "'A' gai wak 2 (1987"
: chr "'A Bit' Too Much Too Soon (1983"
: chr "'A Fish Story' (2013"
: chr "'A Legge (1920"
: chr "'A mala nova (1920"
: chr "'A peggio offesa (1924"
: chr "'A Santanotte (1922"
: chr "'Akasaka no shimai' yori: yoru no hada (1960"
: chr "'Allo 'Allo! at the London Palladium (1988"
: chr "'Ape (2012"
: chr "'Arriet's Baby (1913"
: chr "'Ave You Got a Male Assistant Please Miss? (1973"
: chr "'B' Girl Rhapsody (1952"
: chr "'Babicky dob\xedjejte presne!' (1984"
: chr "'Bayolente' (1999"
```

We used 'strsplit' command and sep ')'. The character ')' is after the movie year. After reduce additional parts, we can reduce the duplicate movies.

2. Construct a weighted directed graph G(V, E) from the list, while

$$V = \{\text{all actors/actressess in list}\}$$
$$S_i = \{m | i \in V, \text{m is a movie in which i has acted}\}$$
$$E = \{(i,j) | i,j \in V, S_i \cap S_j \neq \phi\}$$

and for each directed Edge i → j, a weight is assigned as |Si∩Sj/|Si|.

[Code]

```
# Edge List Creating

if (exists("part_variable") && exists("tmp_part")) {
   assign(part_variable,tmp_part)
}

count_all_function <- function(start_row,end_row,row_length) {
   if (end_row < start_row) {
      0
   } else {
      (end_row - start_row +1) * row_length - sum(start_row:end_row)
   }

}

r_intersect_function <- function(list1,list2) {
   #Reduce(intersect,list(list1,list2))
   intersect(list1,list2)
}


part_start <- 11
part_end <- 20

for (part in part_start:part_end) {
   part_variable <- paste0("edge_part",part)
   part_variable_name <- part_variable
   if (exists(part_variable_name)) {
```

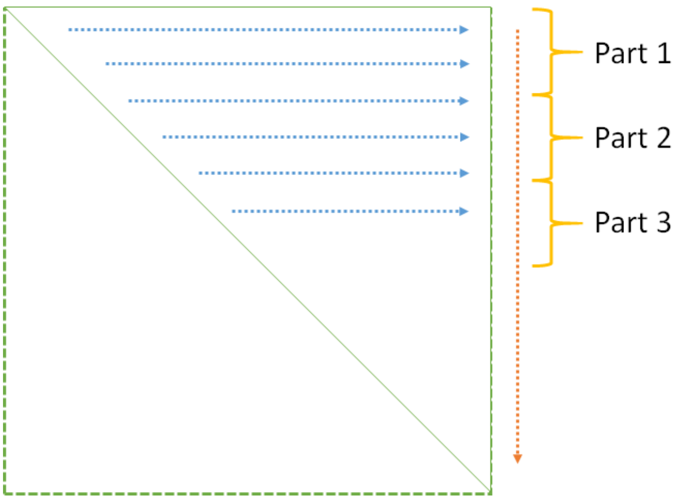
```
        part_start <- part
    } else {
        if (part == part_start) {
            part_start <- part
        } else {
            part_start <- part - 1
        }

        break
    }
}



for (part in part_start:part_end) {
    #part <- 1
    start_row <- ((part-1) * 1000) + 1
    if (part * 1000 > length(extracted_name_list)) {
        # Last Part
        end_row <- length(extracted_name_list)
    } else {
        end_row <- part * 1000
    }

    part_start_row <- start_row
    part_end_row <- end_row
    total_count <- count_all_function(part_start_row,part_end_row,length(extracted_name_list))




    part_variable <- paste0("edge_part",part)

    tmp_part <- list()

    continue <- 0


    part_variable_name <- part_variable



    if (exists(part_variable_name)) {

        continue <- 1

        tmp_part <- get(part_variable_name)
        start_row <- as.numeric(which(extracted_name_index_list==tmp_part[[length(tmp_part)]][1]))
        start_col <- as.numeric(which(extracted_name_index_list==tmp_part[[length(tmp_part)]][2]))

        if (start_row < start_col) {
            # Incomplete finish -> Erase it and continue
            tmp_part[[length(tmp_part)]] <- NULL
            start_row <- as.numeric(which(extracted_name_index_list==tmp_part[[length(tmp_part)]][2]))
            start_col <- as.numeric(which(extracted_name_index_list==tmp_part[[length(tmp_part)]][1]))
        } else {
```

```
        # Complete finish -> Continue to next
        start_row <- as.numeric(which(extracted_name_index_list==tmp_part[[length(tmp_part)]][2]))
        start_col <- as.numeric(which(extracted_name_index_list==tmp_part[[length(tmp_part)]][1]))
        if (start_col == length(extracted_name_list)) {
          # End of Row -> Go to Next Row
          start_row <- start_row + 1
          start_col <- start_row + 1
        } else {
          # Current Row Continue
          start_col <- start_col + 1
        }
      }

  } else {
      #assign(part_variable,list())
      start_row <- part_start_row
      start_col <- start_row + 1
  }

  if (continue == 1) {
      current_count <- count_all_function(part_start_row,(start_row-1),length(extracted_name_list)) +
(start_col-start_row)
  } else {
      current_count <- 1
  }


  percent <- 0
  for (i in start_row:end_row) {
      from_index <- extracted_name_index_list[[i]]
      if (continue == 1) {
        # Continue Work
        for (j in start_col:length(extracted_name_list)) {
          intersect_list <- r_intersect_function(extracted_movie_list[[i]],extracted_movie_list[[j]])
          if (length(intersect_list) > 0) {
            to_index <- extracted_name_index_list[[j]]

            tmp_part <-
append(tmp_part,list(c(from_index,to_index,length(intersect_list)/length(extracted_movie_list[[i]])),c(to_index
,from_index,length(intersect_list)/length(extracted_movie_list[[j]]))))
          }
          #rm(intersect_list)
          current_count <- current_count + 1
        }
      } else {
        # Work to New Row
        for (j in (start_row+1):length(extracted_name_list)) {
          intersect_list <- r_intersect_function(extracted_movie_list[[i]],extracted_movie_list[[j]])
          if (length(intersect_list) > 0) {
            to_index <- extracted_name_index_list[[j]]

            tmp_part <-
append(tmp_part,list(c(from_index,to_index,length(intersect_list)/length(extracted_movie_list[[i]])),c(to_index
,from_index,length(intersect_list)/length(extracted_movie_list[[j]]))))
          }
          #rm(intersect_list)
```

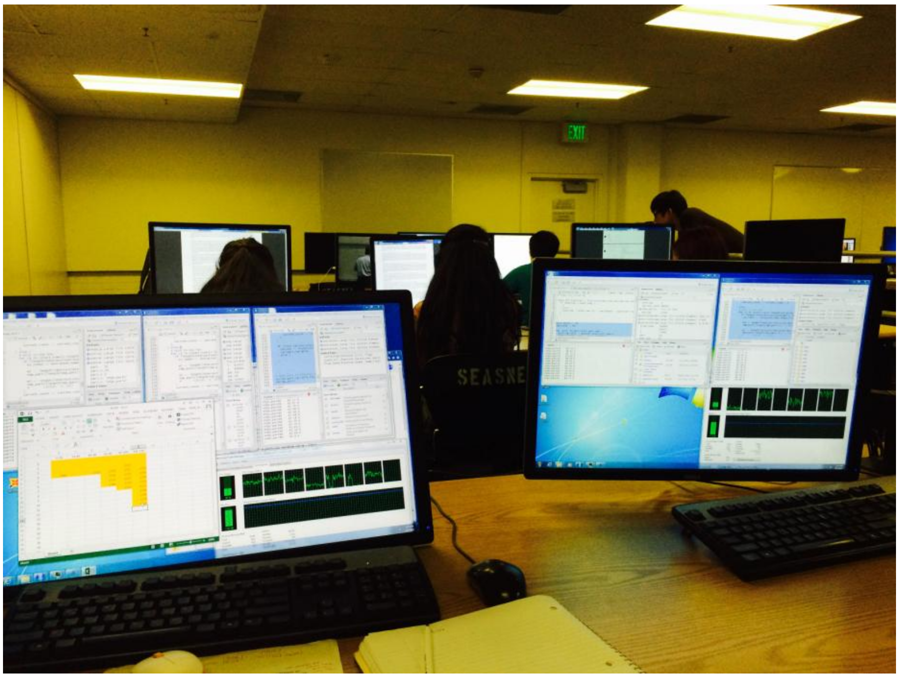
```
            current_count <- current_count + 1
         }
      }

      if (trunc(current_count/total_count*10000) > percent) {
         percent <- trunc(current_count/total_count*10000)
         cat(part_variable,'    ',(percent/100),'%','\n')
      } else {

      }
   }



   # When Part is Complete Save to the Part List
   assign(part_variable,tmp_part)
   #rm(tmp_part)
}
```

The second part is the most difficult part. There were tremendous data to process. The graph is a directed one. Therefore, we need N^2 probable numbers of edges. The extracted number from part 1 with threshold 10 was 113124. Therefore, we need to iterate approximately 113124^2=12797039376. So we need to reduce the iteration number of processing. We have to calculate two edge weights for same two nodes by the direction. But there is no need to calculate weights independently. What I mean here is that when we process some start node and end node we can calculate both directional weights just one time. Therefore, we do not need to iterate all N^2 numbers.

As you see in the below Figure, we will iterate upper trigonal matrix part. However there still a problem that there are gigantic amount of data.



We tried to create edgelist in one program. However, the data accumulates exponentially so there were errors and failure. Therefore, we designed to create edgelist by seperating the parts. Each part contains 1000 rows of the upper tri-diagonal matrix. We had 113124 rows for process, so total number of parts was 120.



We operated the code to multiple PCs and also operated multiple 'R studio' in one PC. The total time for creating all edge list was 8 hours with 3 PCs.

```
edge_list          Large list (22633444 elements, 1.7 Gb)
```

The total number of edges was 22633444 and 1.7Gb size.

[Code]

```
# Combining Edge Files
edge_list <- list()
for (part in 1:114) {
   part_variable <- paste0("edge_part",part)
   tmp_part <- get(part_variable)
   edge_list <- append(edge_list,tmp_part)
   cat('Part',part,' Combining\n')
   if (part == 114) {
      cat('Complete\n')
   }
   rm(tmp_part)
}

for (part in 1:114) {
   part_variable <- paste0("edge_part",part)
   eval(parse(text=paste0("rm(",part_variable,")")))
}
rm(part)
rm(part_variable)



# Creating Edge List Text File
sink(file = "c:/down/edge_list.txt", append = FALSE)
writeLines(unlist(lapply(edge_list, paste, collapse="\t")))
sink()
```

```
# Create Graph
g <- read.graph("C:/down/edge_list.txt", directed=TRUE, format="ncol")
length(E(g))
length(V(g))
```

```
> g <- read.graph("C:/down/
col")
> length(V(g))
[1] 112736
> length(E(g))
[1] 22217037
```

The created graph had 112736 vertices and 22217037 edges.

3. Run pagerank algorithm on the actor/actress network, look into those who are among top 10, do you know their names? List the top 10 famous movie celebrities in your opinion, what are their pagerank scores? Do you see any siginificant pairings amongst actors? Any major surprises, in the sense that well-known actors do not show up in the high pagerank list?

[Code]

```
1   Flowers, Bess   0.000253435  Movies: 828
2   Tatasciore, Fred   0.0001882564  Movies: 355
3   Miller, Harold (I)   0.0001862914  Movies: 561
4   Jeremy, Ron   0.000177933  Movies: 637
5   Lowenthal, Yuri   0.0001616252  Movies: 318
6   O'Connor, Frank (I)   0.000152979  Movies: 623
7   Conaty, James   0.0001488091  Movies: 398
8   Steers, Larry   0.0001468601  Movies: 546
9   Sayre, Jeffrey   0.0001465645  Movies: 430
10   Downes, Robin Atkin   0.0001457329  Movies: 267
```

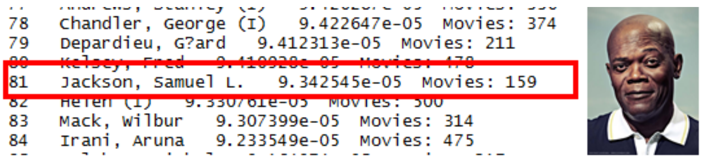
We extracted top 10 actor/actress by Pagerank algorithm. The highest probability was 0.000253435. In the list, we couldn't find who is very famous nowadays. Top 10 actor/actresses have a common point that they have a lot of movies. The top 1 actor 'Flowers, Bess' had 828 movies. We guess that these large amount of movies may affect to the pagerank algorithm. This is because pagerank algorithm based on the random walk in the network. So the random walker must visit frequently who connected to others with many edges. So we tried to find some actor/actresses we know.

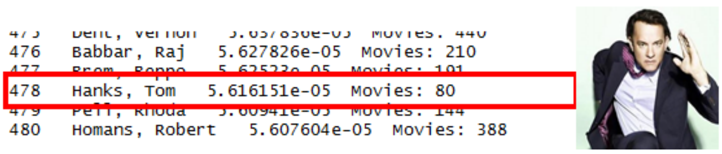
Robert Downey Jr. had 90 movies and page rank was 4.83e-5. And Vin Diesel had 38 movies and the pagerank was 1.94e-5.

```
1   Flowers, Bess    0.000253435  Movies: 828
2   Tatasciore, Fred    0.0001882564  Movies: 355
3   Miller, Harold (I)    0.0001862914  Movies: 561
4   Jeremy, Ron    0.000177933  Movies: 637
5   Lowenthal, Yuri    0.0001616252  Movies: 318
6   O'Connor, Frank (I)    0.000152979  Movies: 623
7   Conaty, James    0.0001488091  Movies: 398
8   Steers, Larry    0.0001468601  Movies: 546
9   Sayre, Jeffrey    0.0001465645  Movies: 430
10  Downes, Robin Atkin    0.0001457329  Movies: 267
11  North, Nolan    0.0001443986  Movies: 227
12  Sullivan, Charles (I)    0.0001427832  Movies: 512
13  Kemp, Kenner G.    0.0001412145  Movies: 420
14  Brahmanandam    0.0001401336  Movies: 973
15  Hamilton, Chuck (I)    0.0001392092  Movies: 414
16  Magrill, George    0.0001335524  Movies: 435
17  Roberts, Eric (I)    0.0001325198  Movies: 298
18  Mower, Jack    0.000128969  Movies: 593
19  O'Brien, William H.    0.000128916  Movies: 437
20  Vogan, Emmett    0.0001287798  Movies: 496
21  Flavin, James    0.0001278372  Movies: 407
22  Dorr, Lester    0.0001270182  Movies: 440
23  Brooks, Ralph    0.0001259659  Movies: 367
24  Strang, Harry    0.0001248793  Movies: 483
25  Ring, Cyril    0.0001247241  Movies: 409
26  Birbal    0.0001222428  Movies: 429
27  Kapoor, Shakti    0.0001210115  Movies: 618
28  Stevens, Bert (I)    0.0001201968  Movies: 337
29  Diaz, Paquito    0.0001194086  Movies: 486
30  Hagney, Frank    0.0001185444  Movies: 404
31  Moorhouse, Bert    0.0001185195  Movies: 352
32  Bailey, Laura (II)    0.0001170791  Movies: 195
33  O'Brien, Liam (V)    0.0001161023  Movies: 220
34  Mills, Frank (I)    0.0001142212  Movies: 363
35  Sherlock, Charles (I)    0.0001142185  Movies: 328
36  Trejo, Danny    0.000113808  Movies: 241
37  Hack, Herman    0.0001135792  Movies: 668
38  Bacon, Irving    0.0001135777  Movies: 510
39  Baker, Troy (II)    0.0001131082  Movies: 159
40  Thornton, Kirk    0.0001128883  Movies: 231
41  Chefe, Jack    0.0001127209  Movies: 336
42  Asrani    0.0001115387  Movies: 406
43  Harris, Sam (II)    0.0001105448  Movies: 600
44  Wahlgren, Kari    0.0001093746  Movies: 179
45  Corrado, Gino    0.0001069374  Movies: 408
```

```
56  Sullivan, Brick    0.000101655  Movies: 319
57  Garcia, Eddie (I)    0.0001016449  Movies: 550
58  Benedict, Brooks    0.0001015794  Movies: 324
59  Willingham, Travis    0.0001015113  Movies: 148
60  Boteler, Wade    0.0001010063  Movies: 448
61  Byron, Tom (I)    0.0001008229  Movies: 386
62  Jackson, Selmer    0.0001007439  Movies: 401
63  Kaufman, Lloyd    0.0001004091  Movies: 301
64  Riehle, Richard    0.0001000068  Movies: 197
65  Wittenberg, Dave (I)    9.777536e-05  Movies: 181
66  Mortimer, Edmund    9.75238e-05  Movies: 336
67  Phelps, Lee (I)    9.703408e-05  Movies: 647
68  Chopra, Prem    9.662222e-05  Movies: 338
69  McCullough, Philo    9.62119e-05  Movies: 378
70  Conklin, Heinie    9.542338e-05  Movies: 479
71  Chandler, Lane    9.535523e-05  Movies: 348
72  Chandler, Eddy    9.517389e-05  Movies: 346
73  Emery, Gideon (I)    9.510741e-05  Movies: 129
74  O'Malley, Pat (I)    9.462442e-05  Movies: 404
75  Kenny, Colin    9.430701e-05  Movies: 263
76  Foulger, Byron    9.427904e-05  Movies: 362
77  Andrews, Stanley (I)    9.426287e-05  Movies: 336
78  Chandler, George (I)    9.422647e-05  Movies: 374
79  Depardieu, G?ard    9.412313e-05  Movies: 211
80  Kelsey, Fred    9.410928e-05  Movies: 478
81  Jackson, Samuel L.    9.342545e-05  Movies: 159
82  Helen (I)    9.330761e-05  Movies: 500
83  Mack, Wilbur    9.307399e-05  Movies: 314
84  Irani, Aruna    9.233549e-05  Movies: 475
85  Galabru, Michel    9.164974e-05  Movies: 217
86  Epcar, Richard    9.154559e-05  Movies: 180
87  Blystone, Stanley    9.0747e-05  Movies: 536
88  Lee, Christopher (I)    9.017344e-05  Movies: 249
89  Ellis, Frank (I)    9.01479e-05  Movies: 557
90  Strong, Tara    9.011747e-05  Movies: 174
91  Madsen, Michael (I)    8.972719e-05  Movies: 218
92  Silvera, Joey    8.962061e-05  Movies: 400
93  Gough, Michael (II)    8.954327e-05  Movies: 151
94  Hearn, Edward    8.952338e-05  Movies: 396
95  Brandenburg, Chet    8.944414e-05  Movies: 238
96  Freeman, Crispin    8.897432e-05  Movies: 149
```

We tried to find the famous actor/actress who we know in the pagerank result.





We found Samuel Jackson on 81, and Tom Hanks on 478. We felt that the ranking of actor/actress who are famous nowadays was very low. But that might not be true because the graph and pagerank was simulated with one hundred thousand actor/actresses. The pagerank value of 10^{-5} can be determined by famous and high ranker. The percentage of total number is 0.0718% and 0.424%. They are included in a very high group.

For finding a significant parings amongst actors, we thought the weight value is the key of the paring. If some actors or actresses played movies together frequently, their edge weight value must be high. So we sorted the edge matrix by the weight value.

	V1	V2	V3
1	36565	614255	1
2	1589635	43273	1
3	51629	1945634	1
4	59328	280253	1
5	70740	1725169	1
6	70740	1973699	1
7	75000	677497	1
8	1380214	85690	1
9	808477	87071	1
10	3053723	87482	1
11	1234671	95600	1
12	96197	223483	1
13	101997	1750755	1
14	1774603	114909	1
15	2143527	114909	1
16	119683	706372	1
17	119683	839145	1
18	119683	2105455	1
19	119683	2616343	1

```
1  Allen, Chesney     Flanagan, Bud
2  Ranchinho     Alvarenga
3  Anderson, Bud Jerome     Todd, Harry
4  Anghel, David     Calder?, Emilio Janhunen
5  Argame, Boyet     Sayson, Raquel
6  Argame, Boyet     Tupaz, Rene
7  Arnold, Eric (XII)     Gattis, David (II)
8  Nakamura, Hayato     Atsumi, Kiyoshi
9  Hawkes, Lionel     Audreson, Michael
10  Rhodes, Alice O'Connor     August, Edwin
11  Mason, Sully     Babbitt, Harry
12  Babu (XLV)     Brahmanandam
13  Bailey, Marvin     Seckler, Bill
14  Shengwu, Zhang     Bao, Guirong
15  Yuanyi, Ding     Bao, Guirong
16  Barker, Jason (IX)     Glut, Donald F.
17  Barker, Jason (IX)     Hill, Bradford (II)
18  Barker, Jason (IX)     Winckler, William
19  Barker, Jason (IX)     Herington, Marieve
20  Fox, Douglas (IV)     Barty, Billy
21  Collyer, Bud     Beck, Jackson (I)
22  Alexander, Joan (I)     Beck, Jackson (I)
23  Buckelew, Alvin     Beckett, Scotty
24  Bellamy, Thomas     Butterworth Jr., Ernest
```

The sorted matrix was not expected. We expected that the weight value should below than 1. But there were 267 number of edges which has weight value '1'. The weight '1' means that every movie is a subset of others or exactly same with other actor/actress.

4. Similarly, remove all movies with less than 5 actors/actresses on list, construct a movie network according to the set of actors/actresses, with weight assigned as the jaccard index of the actor sets of 2 movies. Now we have an undirected network instead.

[Code]

```
# Part4 - Prepare for Creating Edge List by Movie

# Prepare Movie List for Part4
unlist_movie <- unlist(movie_list)
# Use Unique List to Index
movie_index_list <- unique(unlist_movie)
movie_index_list <- as.character(movie_index_list)
movie_index_list <- movie_index_list[c(-(which(movie_index_list=="")))]
sort_order <-    sort.list(movie_index_list, decreasing=FALSE, method=c("shell"))
movie_index_list <- as.list(rbind(movie_index_list)[,sort_order])

movie_index_data_frame <- as.data.frame(table(movie_index_list)) # Sorted
movie_index_list <- as.list(as.character(movie_index_data_frame[,1]))




duplicate_unlist_movie <- unlist_movie[duplicated(unlist_movie)]
unique_duplicate_movie <- unique(duplicate_unlist_movie)
count_duplicate_movie <- list()
```

```
total_number <- length(unique_duplicate_movie)

count_duplicate_movie <- as.data.frame(table(duplicate_unlist_movie))
movie_threshold <- 10
movie_nodes <- subset(count_duplicate_movie, Freq > movie_threshold)
# 1st Row was 'NA' -> Remove
movie_nodes <- movie_nodes[c(-1),]



# Get Movie Genre File
genre_list <- list()
conn_genre=file("C:/project_2_data/movie_genre.txt",open="r")
genre_list <- strsplit(readLines(conn_genre),"\t\t")
close(conn_genre)

mat_genre <- do.call(rbind, genre_list)
genre_data_frame <- as.data.frame(mat_genre)

# Remove NA Row
genre_data_frame <- genre_data_frame[complete.cases(genre_data_frame),]
# movie_index_list <- as.list(as.character(genre_data_frame[,1]))
genre_list <- as.list(as.character(genre_data_frame[,2]))

# Get Unique Genre -> Use Index
genre_index_list <- unique(genre_list)



# Make Basic Movie List
p4_movie_list <- as.list(as.character(movie_nodes[,1]))
p4_movie_freq_list <-    as.list(as.character(movie_nodes[,2]))



#movie_index_list2 <- movie_index_list[1:100]
#movie_index_list2 <- sort(movie_index_list)



start_movie_index <- 1
j <- 1
if (exists("p4_movie_index_list")) {
   start_movie_index <- length(p4_movie_index_list) + 1
   j <- p4_movie_index_list[[length(p4_movie_index_list)]]
} else {
   p4_movie_index_list <- list()
}
which(movie_index_list == p4_movie_list[[1]])

percent <- 0
total_number <- length(p4_movie_list)

for (i in start_movie_index:length(p4_movie_list)) {
   while (TRUE) {
      if (p4_movie_list[[i]] == movie_index_list[[j]]) {
```

```
            p4_movie_list[[i]] <- j
            break
        } else {
            if (j == length(movie_index_list)) {
                cat('No Index Error ',i,'\n')
                break
            }
        }
        j <- j + 1
    }

    if (trunc(i/total_number*10000) > percent) {
        percent <- trunc(i/total_number*10000)
        cat((percent/100),'%','\n')
    } else {

    }
}
p4_movie_list[[1]]
```

At first, we created unique movie lists. We need to identify the movie with the identical number. So we created the movie index list by the unique movie list. And then we created duplicated movie lists. We need to extract movies which have more than 5 actor/actresses. We set the threshold as a 10 same to the previous actor/actress case. From duplicate movie lists, we made a table matrix by 'table' function. The function counts which elements repeated and creates the table with the frequencies.

	row.names	duplicate_unlist_movie	Freq
1	9	'38 (1986)	29
2	16	'51 Dons (2014)	31
3	21	'70 Remembering a Revolution (2010)	25
4	23	'71 (2014)	72
5	26	'77 (2007)	49
6	27	'77 (2007) (uncredited)	11
7	28	'79 Parts (2015)	143
8	33	'93 jie tou ba wang (1993)	17
9	37	'A' gai wak (1983)	32
10	38	'A' gai wak (1983)	17
11	41	'A' gai wak 2 (1987)	30
12	45	'A Bit' Too Much Too Soon (1983)	13
13	46	'A Fish Story' (2013)	15
14	51	'Akasaka no shimai' yori: yoru no hada (1960)	11
15	59	'Babicky dob�jejte presne!' (1984)	17
16	60	'Bayolente' (1999)	11
17	63	'Breaker' Morant (1980)	39
18	72	'Ch�' kowai hanashi the movie: yami no eigasai (2005)	11
19	74	'Como M�xico no hay dos'! (1945)	11

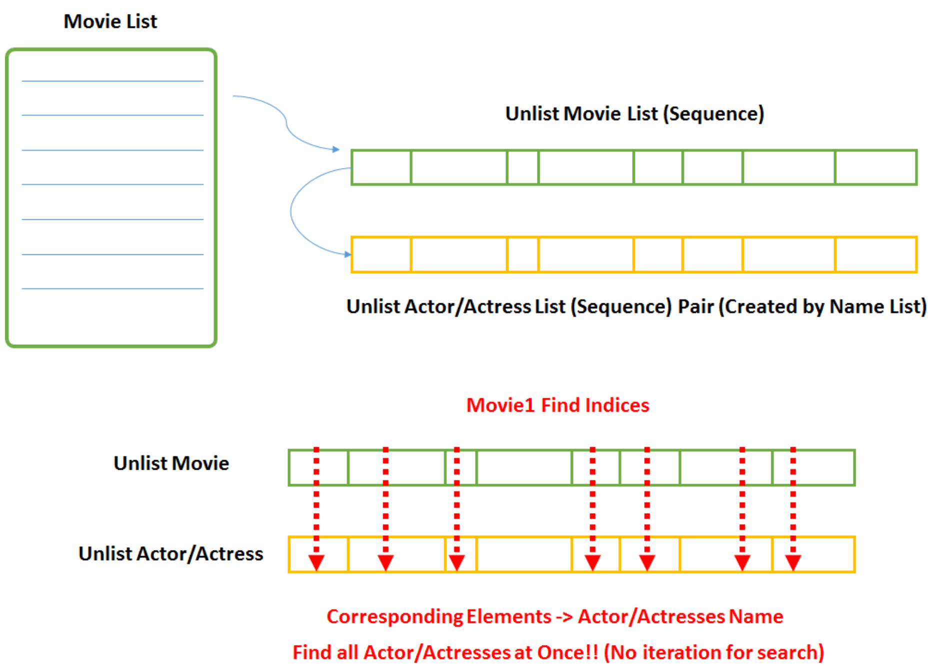
The matrix is created by 'table' function. And we easily created the list which has more than 10 actor/actresses by using 'subset' function. We prepared movie nodes which is satisfied the threshold condition. And the number of movie nodes were 178711. After this, we need an Actor/Actress list for a movie. In previous part 1-3, there are movie lists of each actor or actress. Similar to this we prepare the actor/actress list for a movie.

[Code]

```
# Part4 - 2 Create Act List
#
# Create Act Unlist Matching to Unlist Movie
movie_act_list <- movie_list
percent <- 0
total_number <- length(movie_act_list)
for (i in 1:length(movie_act_list)) {
    name <- name_list[[i]]
    for (j in 1:length(movie_act_list[[i]])) {
        movie_act_list[[i]][j] <- name
    }
    if (trunc(i/total_number*10000) > percent) {
        percent <- trunc(i/total_number*10000)
        cat((percent/100),'%','\n')
    }
}
#rm(unlist_movie_act)
unlist_movie_act <- unlist(movie_act_list)


#unlist_movie_act[unlist_indices]


# Create P4 Actor/Actress List
if (exists("p4_movie_act_list")) {
    if (min(which(p4_movie_act_list=="NULL")) == 1) {
        start_row <- 1
    } else {
        start_row <- min(which(p4_movie_act_list=="NULL")) - 1
    }

} else {
    p4_movie_act_list <- vector("list", length(p4_movie_list))
    start_row <- 1
}



percent <- 0
total_number <- length(p4_movie_list)
start_row <- i
for (i in start_row:71485) {

    p4_movie_act_list[[i]] <- unlist_movie_act[which(unlist_movie == p4_movie_list[[i]])]

    #gc()
    if (trunc(i/total_number*10000) > percent) {
        percent <- trunc(i/total_number*10000)
        cat((percent/100),'%','\t',i,'\n')
    }
}

start_row <- i
```

```
#1-71485

save(i,p4_movie_act_list,file="Actfrom1.RData")
```

It may take long time to find actors or actress one by one. But we found the best way to find the actor or actresses at once.



This is how we find the actor and actresses names not searching one by one. We made unlist movie list first. And we created unlist actor actress name vector by putting repeatedly name on the element. The 'which' function returns indices of element in the vector or list. So we put the same indices to the unlist actor/actress vector. We could finally get the whole names at once.

The next thing is a creating edge list. This process is the same as the previous part. We reused the code for part 2. In this case, the vertices number is 178711. So we totally needed 171 parts.

[Code]

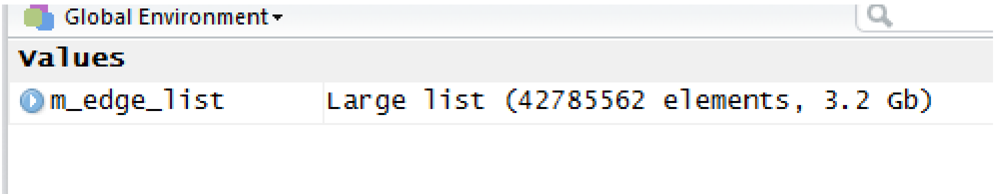
```
# Part 4 - Creating Movie Edges
#
# Movie Edge List Creating

if (exists("part_variable") && exists("tmp_part")) {
   assign(part_variable,tmp_part)
}

count_all_function <- function(start_row,end_row,row_length) {
   if (end_row < start_row) {
      0
   } else {
      (end_row - start_row +1) * row_length - sum(start_row:end_row)
   }

}

r_intersect_function <- function(list1,list2) {
   intersect(list1,list2)
```

```
}

part_start <- 1
part_end <- 10

for (part in part_start:part_end) {
   part_variable <- paste0("movie_edge_part",part)
   part_variable_name <- part_variable
   if (exists(part_variable_name)) {
      part_start <- part
   } else {
      if (part == part_start) {
         part_start <- part
      } else {
         part_start <- part - 1
      }

      break
   }
}



for (part in part_start:part_end) {
   #part <- 1
   start_row <- ((part-1) * 1000) + 1
   if (part * 1000 > length(p4_movie_list)) {
      # Last Part
      end_row <- length(p4_movie_list)
   } else {
      end_row <- part * 1000
   }

   part_start_row <- start_row
   part_end_row <- end_row
   total_count <- count_all_function(part_start_row,part_end_row,length(p4_movie_list))




   part_variable <- paste0("movie_edge_part",part)

   tmp_part <- list()

   continue <- 0


   part_variable_name <- part_variable



   if (exists(part_variable_name)) {

      continue <- 1

      tmp_part <- get(part_variable_name)
```

```
        start_row <- as.numeric(which(p4_movie_index_list==tmp_part[[length(tmp_part)]][1]))
        start_col <- as.numeric(which(p4_movie_index_list==tmp_part[[length(tmp_part)]][2]))


    } else {
        #assign(part_variable,list())
        start_row <- part_start_row
        start_col <- start_row + 1
    }

    if (continue == 1) {
        current_count <- count_all_function(part_start_row,(start_row-1),length(p4_movie_list)) + (start_col-
start_row)
    } else {
        current_count <- 1
    }


    percent <- 0
    for (i in start_row:end_row) {
        from_index <- p4_movie_index_list[[i]]
        if (continue == 1) {
            # Continue Work
            for (j in start_col:length(p4_movie_list)) {
                intersect_list <- r_intersect_function(p4_movie_act_list[[i]],p4_movie_act_list[[j]])
                if (length(intersect_list) > 0) {
                    to_index <- p4_movie_index_list[[j]]


                    tmp_part <-
append(tmp_part,list(c(from_index,to_index,length(intersect_list)/(length(p4_movie_act_list[[i]])+length(p4_
movie_act_list[[j]])-length(intersect_list)))))
                }
                current_count <- current_count + 1
            }
        } else {
            # Work to New Row
            for (j in (start_row+1):length(p4_movie_list)) {
                intersect_list <- r_intersect_function(p4_movie_act_list[[i]],p4_movie_act_list[[j]])
                if (length(intersect_list) > 0) {
                    to_index <- p4_movie_index_list[[j]]

                    tmp_part <-
append(tmp_part,list(c(from_index,to_index,length(intersect_list)/(length(p4_movie_act_list[[i]])+length(p4_
movie_act_list[[j]])-length(intersect_list)))))
                }
                current_count <- current_count + 1
            }
        }

        if (trunc(current_count/total_count*10000) > percent) {
            percent <- trunc(current_count/total_count*10000)
            cat(part_variable,'    ',(percent/100),'%','\n')
        } else {

        }
```

```
    }


    # When Part is Complete Save to the Part List
    assign(part_variable,tmp_part)
}
```

We repeatedly operated the code for creating edges. And we got 171 parts edge lists. It included 42785562 edges and 3.2Gb size.

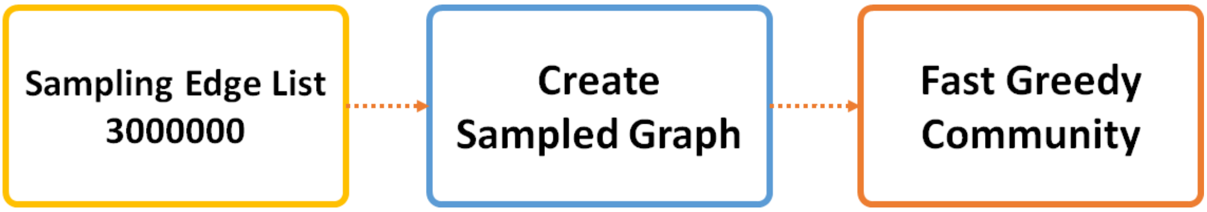


[Code]

```
# Part 5

# Edge List Was too Big -> Sampling!!
sample_movie_edge_3000000 <- sample(m_edge_list,3000000)

sink(file = "c:/down/0612 Work/sample_edge_list_3000000.txt", append = FALSE)
writeLines(unlist(lapply(sample_movie_edge_3000000, paste, collapse="\t")))
sink()
rm(sample_movie_edge_3000000)


g <- read.graph("c:/down/0612 Work/sample_edge_list_3000000.txt", directed=FALSE, format="ncol")

g <- simplify(g, remove.multiple = TRUE, remove.loops = TRUE, edge.attr.comb = getIgraphOpt("edge.attr.comb"))
is.simple(g)
fg <- fastgreedy.community(g)

length(V(g))
length(E(g))

for (i in 1:length(genre_movie_list)) {
    genre_movie_list[[i]] <- strsplit(genre_movie_list[[i]],")")[[1]]
}


# ith Community members index return
#as.numeric(fg$names[which(fg$membership == i)])
#unlist(movie_index_list[as.numeric(fg$names[which(fg$membership == i)])])
#unlist(genre_number_list[which(unlist(movie_index_list[as.numeric(fg$names[which(fg$membership == i)])]) %in% genre_movie_list)])
```

```
genre_number_list <- vector("list", length(genre_list))
for (i in 1:length(genre_list)) {
   genre_number_list[[i]] <- which(genre_index_list %in% genre_list[[i]])
}


# Find communities which sizes are over 100
length(which(sizes(fg) > 100))
#which(sizes(fg) > 100)
sizes(fg)[which(sizes(fg) > 100)]

# Get community number from the Biggest size to lower
community_order_list <- list()
for (i in 1:length(which(sizes(fg) > 100))) {
   community_order_list[[i]] <- which(sizes(fg) == sort(sizes(fg)[which(sizes(fg) > 100)],decreasing=TRUE)[[i]])[[1]]
}


community_genre_list <- list()
for (i in 1:length(which(sizes(fg) > 100))) {
   community_genre_list[[i]] <- unlist(genre_number_list[which(unlist(movie_index_list[as.numeric(fg$names[which(fg$membership == community_order_list[[i]])])]) %in% genre_movie_list)])
}

for (i in 1:length(which(sizes(fg) > 100))) {
   hist(community_genre_list[[i]],29)
   hist(community_genre_list[[i]][which(community_genre_list[[i]] != 4)],29)
}
```

5. Do a community finding on the movie network; use the Fast Greedy Newman algorithm. Tag each community with the genres that appear in 20% or more of the movies in the community. Are these tags meaningful?



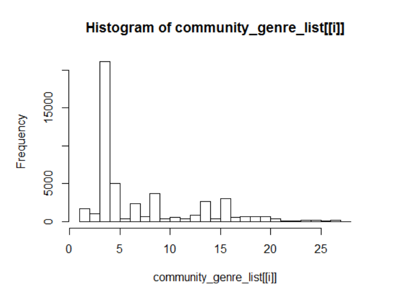
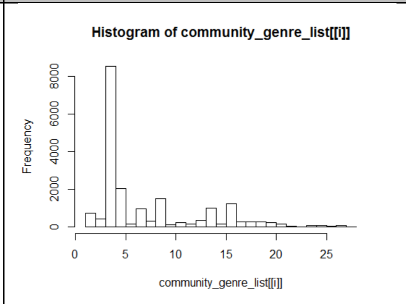
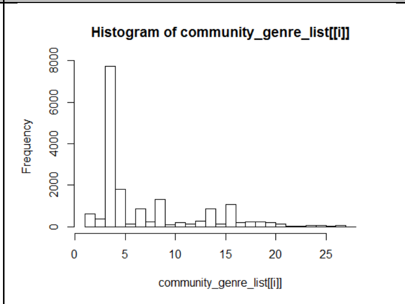
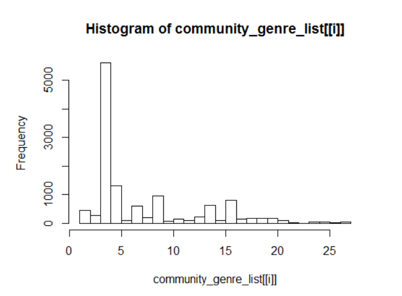
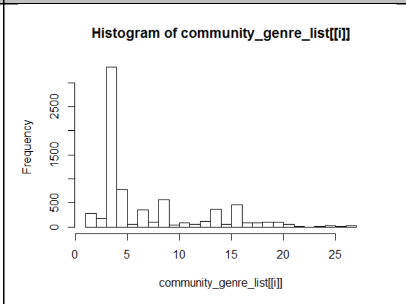
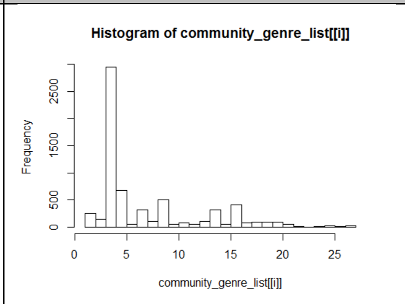
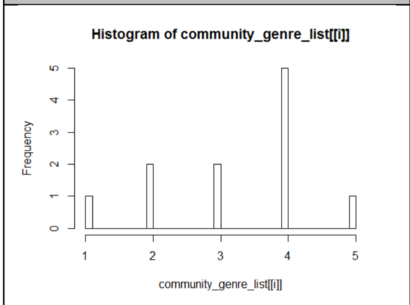
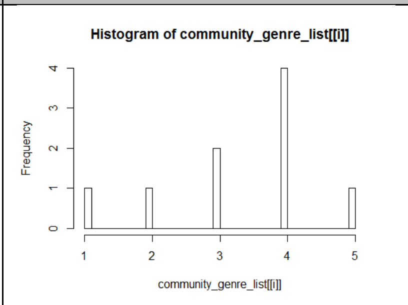
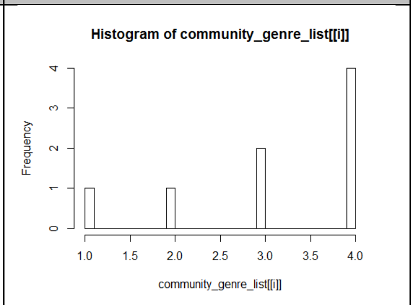
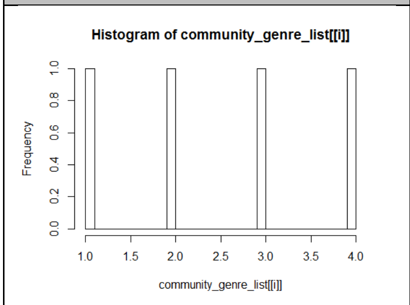
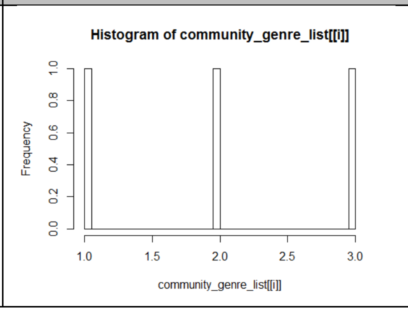
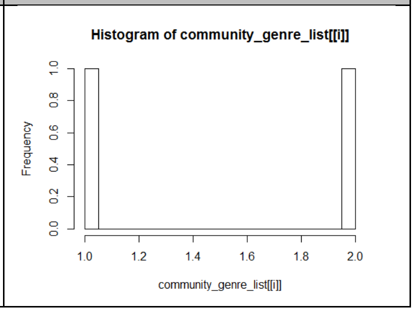
We tried to make a graph directly from the movie edge list. But it was failed because of memoryless. So we decided to sample a part of entire edge list. We took 3000000 sample edges and created the graph.

```
> length(V(g))
[1] 166675
> length(E(g))
[1] 2986774
```

The graph was created. The number of vertices was 166675 and the number of edges was 2986774. The sampling number was less than 10% of whole edge list. But the vertex number was different not much. And we tried to find the community by ‘Fast Greedy’ algorithm.

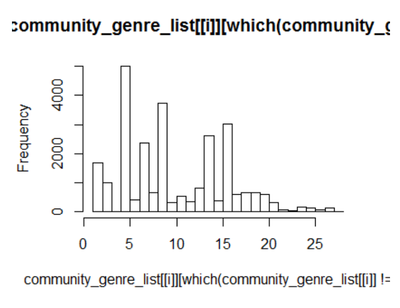
```
> length(which(sizes(fg) > 100))
[1] 25
> #which(sizes(fg) > 100)
> sizes(fg)[which(sizes(fg) > 100)]
Community sizes
    1     2     3     4     5     6     7     8     9    10    11    12    13
48455  6946 21928 12707  5182  2397  5267 17234  1299  9581  5267  1929  4050
  115
  281
```

We searched how many communities are in the graph which number is over 100. There were 25 communities. And then we tagged the genre by each vertex. After completing genre tagging. We histogram 25 communities' genre.

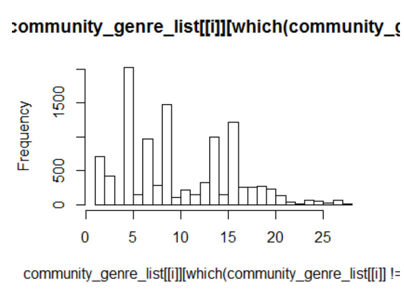
1	2	3
		
4	**5**	**6**
		
24	**25**	**26**
		
31	**32**	**40**
		

When we histogram, the genre index 4 which is "Short" is dominating all the histogram. So we tried to histogram without the genre index 4.

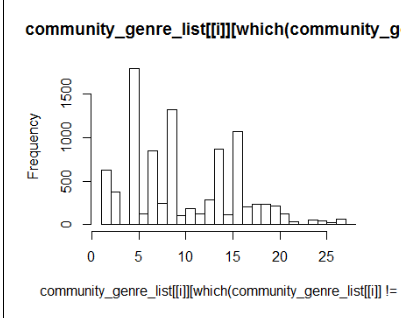
1



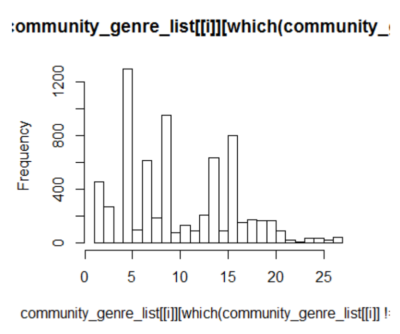
2



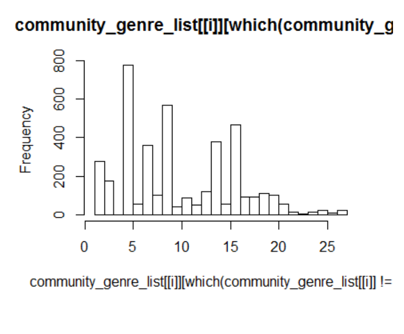
3



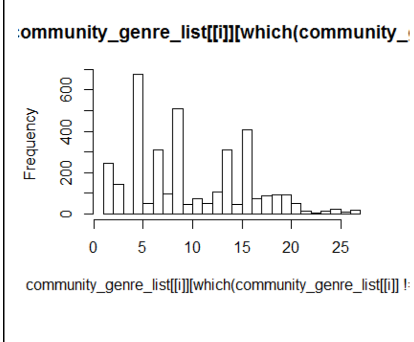
4



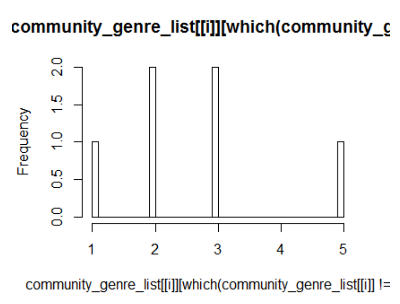
5



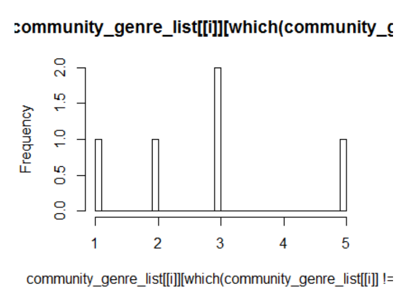
6



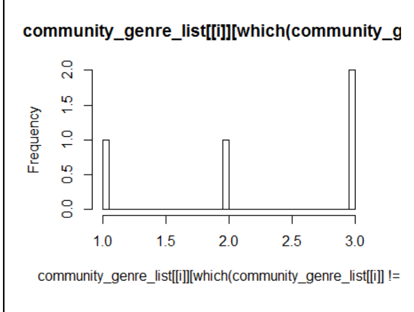
24



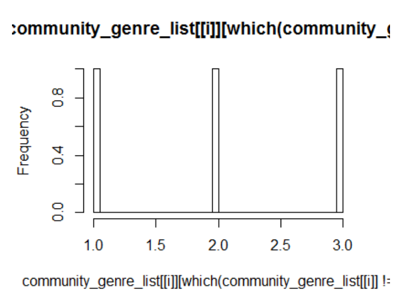
25



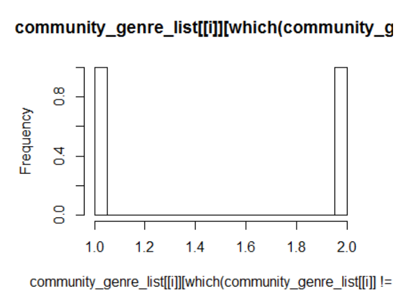
26



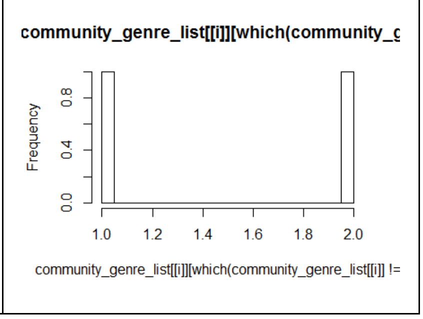
31



32



40



Without index4, the next determining genre was number 5, “Drama” and also number 9 “Documentary” was high. And in low number of communities were tagged 1, 2, or 3. We expected that the community may show different genre distribution. However, almost every communities shows similar genre distribution. It might be caused by many data. Many vertices were connected each other complex so it’s hard to separate so the genre could not be deterministic factor.

6. Add the following nodes into the network, For each of them, return the top 5 nearest neighbors. Which communities does each of them belong to?

Batman v Superman: Dawn of Justice (2016)

Mission: Impossible - Rogue Nation (2015)

Minions (2015)

[Code]

Before adding vertices to the graph, we found entire neighbors of each movie sharing actor or actress in whole data. I found 375 neighbors for 'Batman v Superman', 346 neighbors for 'Mission Impossible', and 468 for 'Minions'.

In sampled network, there were 44 neighbors for 'Batman v Superman', 30 neighbors for 'Mission Impossible', and 51 neighbors for 'Minions'.

```
1   A Noite e a Madrugada (1986   Community: 3
2   Best Laid Plans (1999   Community: 1
3   Bettie Page Reveals All (2012   Community: 1
4   A Man Called Gannon (1968   Community: 3
5   Ana Maria in Novela Land (2015   Community: 1
6   Ana Maria in Novela Land (2015   Community: 1
7   Alpengl?n im Dirndlrock (1974   Community: 10
8   Alpengl?n im Dirndlrock (1974   Community: 10
9   All in Good Time (2012   Community: 4
10   All in Good Time (2012   Community: 4
```

Batman v Superman Nearest Neighbors

Nearest 5 neighbors are belongs to community 1 and community 3. Community 1 were three and community 3 were two. So community 1 may be closer to 'Batman v Superman'.

```
1   Battle of Rogue River (1954   Community: 8
2   Battle of Rogue River (1954   Community: 8
3   Battle of Rogue River (1954   Community: 8
4   Bheja Fry 2 (2011   Community: 5
5   A Grim Tale (2014   Community: 257
6   A Descent of Woodpeckers (2004   Community: 1
7   Auditorium parco della musica: l'officina dell'arte (2013   Community: 356
8   Au suivant! (2000   Community: 3
9   Au suivant! (2000   Community: 3
10   Ai chu se (2010   Community: 7
```

Mission Impossible Rogue Nation Nearest Neighbor

There are some error in the list. First top five elements were duplicated. But there seems no pattern because nearest neighbors belongs to each different community.

```
1   Anime nere (2014   Community: 3
2   Bonhoeffer: Agent of Grace (2000   Community: 10
3   Ala Bala Nica (2011   Community: 122
4   Batad: Sa paang palay (2006   Community: 17
5   Batad: Sa paang palay (2006   Community: 17
6   Bairaag (1976   Community: 5
7   Blokada: Luzhskiy rubezh, Pulkovskiy meredian (1975   Community: 16
8   Blokada: Luzhskiy rubezh, Pulkovskiy meredian (1975   Community: 16
9   Bill - Das absolute Augenmass (2008   Community: 3
10   Batacl? mexicano (1956   Community: 2
```

Minions Nearest Neighbor

In top five community, it is hard to find which community Minions belong to. There were no repeat communities.

```
> fg2$membership[which(fg2$names == as.character(whi
man: Dawn of Justice (2016")))]
[1]    1 1382
> fg2$membership[which(fg2$names == as.character(whi
sible - Rogue Nation (2015")))]
[1]    1 1383
> fg2$membership[which(fg2$names == as.character(whi
)))]
[1]    1 1384
```

We found the result of fast greedy algorithm. The three movies are together belonging to the community 1.

7. Download the ratings list, derive a function to predict the ratings of the above 3 movies using the movie network. (hint: try to use the ratings of neighbor movies and movies in the same community.)

Neighbor-based rating prediction

Firstly, we obtain the rating prediction by using the ratings of all the neighbors (neighbors with threshold constraint) of each of the three movies. The ratings by the neighbors are weighted based on the edge weights. We then average out the rating of the neighbors by their weights to predict the rating of the three movies.

Code:

```
totalweightedratingmission = 0
totaledgeweight = 0
for (i in 1:length(missionneimovies))   #based on all neighbors
{
  indexofrating = which(movie_rating[,1] ==trim.trailing(names(missionneimovies)[i]))
  if(isTRUE(indexofrating>0))
  {
    edgeweight = sortmissionneiweight$x[i]
    totaledgeweight = totaledgeweight + edgeweight
    totalweightedratingmission  = totalweightedratingmission + edgeweight*movie_rating[indexofrating,3]
  }
}
ratingmission = totalweightedratingmission/totaledgeweight
```

Result:

```
> ratingmission
[1] 6.625282
> ratingbatman
[1] 6.871994
> ratingminion
[1] 6.462112
```

Combination of neighbor-based and community-based rating prediction:

We can combine prediction using neighbors' information with rating information from the nodes within the same communities. For example, prediction of rating based on total number of ratings from all the communities each node belongs, and average out by the intensity of the community with respect to that node. Or, we can also use the averages of the average community ratings.

For example, using formula :

Predicted_rating = w1 (neighbor-based prediction) +w2 (community-based prediction)

Where w1 and w2 are [0,1]. This depends on how much we weigh the importance of neighbors vs community in prediction.

8. Using a set of features that include the following:

- **top 5 pageranks of the actors (five floating point values) in each movie.**

- **if the director is one of the top 100 directors or not (101 boolean values). These are directors of the top 100 movies from the "IMDb top 250". You can also find a list of these movies in the ratings.list.gz file.**

train a regression model and predict the ratings of the 3 movies mentioned above. Specify the exact feature set you use and how you compute the numerical values for these features. Compute and state the goodness of fit for your regression model.

Top 100 movies

```
> moviesorted
  [1] "The Fighting Season (2015)"                                  "Jes and Lora (2015)"
  [3] "Mr.Werner Herzog! My Best Friends! Meet at Tamsui! (2007)"   "Tango (2015/II)"
  [5] "Toutai (2015)"                                               "Milf Revolution (2013)"
  [7] "The Brother Load 6 (2014)"                                   "Stop-and-Cop (2009)"
  [9] "Warning Labels (2015)"                                       "Private Specials 3: Bisexual Dreamer (2008)"
 [11] "God Provides (2009)"                                         "Birdsong (2013)"
 [13] "John Mayer: Someday I'll Fly (2014)"                         "Wiedzmin 3: Dziki Gon (2015)"
 [15] "Brotherhood of the Popcorn (2014)"                           "Bedtime (2014)"
 [17] "Rocket Jockey (1996)"                                        "Warum (2015)"
 [19] "Brooklyn Sweet Nothings (2013)"                              "Madly Unto Eternity (2012)"
 [21] "Blowjob Adventures of Dr. Fellatio 23 (2000)"                "The Haun Solo Project: Addicted (2012)"
 [23] "Black Owned 3 (2008)"                                        "Den tha gerasoume pote (2014)"
 [25] "Meester Rob (2014)"                                          "Zombie Cops (2014)"
 [27] "A Halloween Carol (2014)"                                    "The Chemist (2012)"
 [29] "Rocket Beans TV (2012)"                                      "Fear No Fruit (2015)"
 [31] "Godkiller (2015)"                                            "Taz Wanted (2002)"
 [33] "Pop Legends Live: Johnny Maestro & the Brooklyn Bridge (2005)" "Meet Heather (2007)"
 [35] "The Brother Load 2 (2010)"                                   "The Last of Us (2013)"
 [37] "The Cannibal of Paraná (2014)"                               "Anal Devastation 2 (2008)"
 [39] "Girls Loving Girls (1996)"                                   "Token of Love (2015)"
 [41] "The Gift (2014/XIV)"                                         "Big Voice (2015)"
 [43] "Medusa (2015/IV)"                                            "Boob Cruise 2000 (2000)"
 [45] "Eleven (2014/V)"                                             "Fiktion (2014)"
 [47] "Stuffin' Young Muffins 8 (2007)"                             "A Brave Heart: The Lizzie Velasquez Story (2015)"
 [49] "Contrary to Likeness (2014)"                                 "Jalebi (2013)"
 [51] "True Heroes (2014)"                                          "Danica Mature Erotic (2006)"
 [53] "Beneath the Helmet (2014)"                                   "Nopperabou (2015)"
 [55] "Paramore: Misery Business (2007)"                            "Heads and Tails (1999)"
 [57] "2.em (2014)"                                                 "Aleppo. Notatki z ciemnosci (2014)"
 [59] "Big Bust Superstars (1983)"                                  "Dreamer (2015)"
 [61] "Marching to Zion (2015)"                                     "Tape Busters Vol. 1 (1985)"
 [63] "The History of USC Football (2005)"                          "Stealth (2015)"
 [65] "Scouts Honor: Inside a Marching Brotherhood (2014)"          "Naughty Naturals 3 (2004)"
 [67] "Gayze (2015)"                                                "Stamatiste to tragoudi (2014)"
 [69] "Guides (2011)"                                               "A Fanatic by Choice (2015)"
 [71] "Keyhole Productions 214: Christy Canyon Sucks (1989)"        "I Thought I Told You to Shut Up!! (2015)"
 [73] "Tom Clancy SSN (1996)"                                       "P.O.V. Juggfuckers 5 (2013)"
 [75] "One Die Short (2014)"                                        "Metal Gear Solid (1998)"
 [77] "El Abogado Del Diablo (2013)"                                "Irreversible (2015)"
 [79] "Ed Sullivan Presents: Rock 'N Roll Revolution (2011)"        "5 Seconds of Summer: Amnesia (2014)"
 [81] "Erica's Debut (2001)"                                        "Daddy (2013)"
 [83] "The Networker (2015)"                                        "Beverley (2015)"
 [85] "Genocide Gentleman: Class A War Criminals of UK and US (2015)" "Torche: Annihilation Affair (2015)"
 [87] "Still Falls the Rain (2012)"                                 "Cupid Carries a Gun (2014)"
 [89] "The Brother Load (2009)"                                     "Fragile Waters (2014)"
 [91] "Painkiller Already (2010)"                                   "Grand Theft Auto V (2013)"
 [93] "Fake Mustachios (2012)"                                      "Dead Bird Don't Fly (2014)"
 [95] "Dogs of War (2012)"                                          "Hercules' Hero Quest (1998)"
 [97] "Private Movies 44: Fuck TV (2008)"                           "A Hollywood Zone (2011)"
 [99] "Bear Sex Party (1996)"                                       "Details (2014)"
```

Figure 1 Top movies sorted by rating

Top directors

```
> topdirector
  [1] NA                        NA                  NA                      NA                 "Surendra, Saie"   NA
  [7] NA                        NA                  "Morrison, Jennifer (II)" NA               NA                 "Childs, Ben (II)"
 [13] NA                        NA                  "Reid, Inda"            NA                 NA                 NA
 [19] "Hackeling, Patrick"      "Holdredge, Pikey"  NA                      NA                 NA                 NA
 [25] "Mathôt, Jules"           NA                  NA                      NA                 NA                 "Smith, Mark Brian"
 [31] "Hageman, Jordan"         NA                  NA                      NA                 NA                 NA
 [37] "Mastrolorenzo, José Luis" NA                 NA                      NA                 NA                 NA
 [43] NA                        NA                  "Retelas, George"       NA                 NA                 NA
 [49] "Dapp, Motke"             "Hasan, Shajee (I)" "Ganucheau, Chris"      NA                 NA                 NA
 [55] NA                        "Dana, Michael (I)" NA                      NA                 NA                 "Homan, Kyle"
 [61] NA                        NA                  NA                      NA                 "Smith, Mac (I)"   NA
 [67] "Hartley, Tom (III)"      "Manos"             "Tenenbaum, Daniel"     NA                 NA                 NA
 [73] NA                        NA                  "Forcella, Matthew (IV)" NA                NA                 NA
 [79] NA                        NA                  NA                      NA                 ""                 NA
 [85] NA                        NA                  "Santana, Miguel (II)"  NA                 NA                 NA
 [91] NA                        NA                  "Gurst, Al"             "Sporns, Charlie"  NA                 NA
 [97] NA                        "Palacios, Cesar"   NA                      "Mewes, Jason"
```

Figure 2 Top 100 directors

Data set is corrupted with inconsistent spacing in the original file, so it becomes hard to identify directors whose different informations are split in inconsistent ways, such as combinations of ‘return’, ‘/t/t’ and ‘/t’

Here is example of the corrupted data, which causes many top directors to be unidentifiable, but some are still retrievable

<u>Corrupted data</u>

```
> director_movies[105270,]
                 V1 V2                                          V3 V4 V5 V6 V7 V8 V9 V10 V11
105270 Lele, Urvashi    An Interview with the Owl and the Pussycat (2014)
> director_movies[105280,]
                  V1 V2                     V3 V4           V5 V6 V7 V8 V9 V10 V11
105280 Lelli, Luciano    Bang Bang Kid (1967)    Ragan (1968)
> director_movies[105285,]
                 V1 V2                      V3 V4                V5 V6 V7 V8 V9 V10 V11
105285 Lello, Leslie    New Jersey 350 (2014)    Real vs. Reel (2006)
> director_movies[105287,]
                V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11
105287 Lluch, Ramón
> director_movies[105286,]
                     V1 V2                           V3 V4
105286 Lellouche, Gilles    2 minutes 36 de bonheur (1996)

V5
105286 Les infidèles (2012)  (segment 'Las Vegas)\t\tNarco (2004)\t\tPourkoi... passkeu (2002)\t\tZéro un (2003)  (segment Pourkoi... pa
skeu)\nLellouche, Michael\t\tLa coagulation des jours (2010)\nLellouche, Philippe\t\tNos plus belles vacances (2012)\t\tUn prince (presq
e) charmant (2013)\nLellouche, Sophie\t\tDieu, que la nature est bien faite! (1999)\t\tParis-Manhattan (2012)\nLelong, Christian\t\tAgad
z nomade FM (2004)\t\tAmour, sexe et mobylette (2008)\t\tJustice à Agadez (2006)\nLelong, Jean-Marc\t\t\nLelong, Marc\t\tYasmine and the
Sex Models (2009)\nLelouch, Claude\t\t...pour un maillot jaune (1965)\t\t11'09''01 - September 11 (2002)  (segment France)\t\t13 jours e
France (1968)\t\t24 heures d'amant (1964)\t\tAnd Now... Ladies and Gentlemen... (2002)\t\tAttention bandits! (1986)\t\tC'était un rendez
vous (1976)\t\tCes amours-là (2010)\t\tChacun son cinéma ou Ce petit coup au coeur quand la lumière s'éteint et que le film commence (20
7)  (segment Cinéma de Boulevard)\t\tD'un film à l'autre (2011)\t\tHasards ou coïncidences (1998)\t\tHommes, femmes, mode d'emploi (1996
\t\tIl y a des jours... et des lunes (1990)\t\tIran (1971)\t\tItinéraire d'un enfant gâté (1988)\t\tJean-Paul Belmondo (1965)\t\tL'amour
avec des si (1964)\t\tL'aventure, c'est l'aventure (1972)\t\tLa belle histoire (1992)\t\tLa bonne année (1973)\t\tLa femme spectacle (19
4)\t\tLa guerre du silence (1957)\t\tLa vie, l'amour, la mort (1969)\t\tLe bon et les méchants (1976)\t\tLe chat et la souris (1975)\t\t
e courage d'aimer (2005)\t\tLe genre humain - 1ère partie: Les Parisiens (2004)\t\tLe propre de l'homme (1961)\t\tLe voyou (1970)\t\tLes
Bandits manchots (????) {{SUSPENDED}}\t\tLes grands moments (1965)\t\tLes misérables (1995)\t\tLes mécaniciens de l'Armée de l'Air (1958
\t\tLes uns et les autres (1981)\t\tLoin du Vietnam (1967)\t\tLumière et compagnie (1995)\t\tMariage (1974)\t\tPartir, revenir (1985)\t\
Robert et Robert (1978)\t\tRoman de gare (2007)\t\tSalaud, on t'aime. (2014)\t\tSi c'était à refaire (1976)\t\tSmic Smac Smoc (1971)\t\t
he King of Ads (1993)  (segment Contrex Eau Minerale Naturelle commercial)\t\tTout ça... pour ça! (1993)\t\tToute une vie (1974)\t\tUn a
tre homme, une autre chance (1977)\t\tUn coup de foudre normand (1990)  (uncredited)\t\tUn homme et une femme (1966)\t\tUn homme et une
emme, 20 ans déjà (1986)\t\tUn homme qui me plaît (1969)\t\tUn plus une (2015)\t\tUne fille et des fusils (1965)\t\tUne pour toutes (199
)\t\tVisions of Eight (1973)  (segment Losers, The)\t\tViva la vie! (1984)\t\tVivre pour vivre (1967)\t\tÀ nous deux (1979)\t\tÉdith et
arcel (1983)\nLelouch, Martine\t\t\nLelouch, Sachka\t\tPasse le briquet à ton voisin (2013)\nLelouch, Seb\t\t\nLelouch, Simon\t\t7.57 AM
PM (2010)\t\tNous sommes tous des anges (1996)\t\tNous sommes tous des êtres penchés (2013)\nLelouche, Simon\t\t\nLeloudas, Kostas\t\tTo
lavaro tou '21 (1929)\nLeloup, Julien\t\tDes anges (2001)\nLeloup, Jérémie\t\tLe loup (2004)\nLelté, Jean-Jacques\t\tLa quille (2004)\nL
luc, Paul\t\t\nLem, Brenda Joy\t\tOpen Letter: Grasp the Bird's Tail (1992)\nLem, Hann-Shi\t\tPrescient (2015)\nLem, Jason\t\tRunning Re
Lights (2004)\nLem, Steve\t\tBeam me up, Scotty! (1996)\nLema, Antonia\t\tPuzzle (2011)\nLema, Carlos M.\t\tBestia blanca (2013)\nLema,
onzalo\t\tEl ventilador asesino (2009)\nLema, Irene\t\tEl charco azul (2013)\nLema, Samuel (II)\t\tLa Carne Cruda (2013)\nLemagnen, Gaël
t\tBabel, les temps silencieux (2012)\t\tJazz (2011)\nLemaine, Brigitte\t\tRegardez moi, je vous regarde (1999)\t\tTémoins sourds, témoi
```

Figure 3Corrupted Data Directors file have both single and double tab

There are some columns in the data frame which is hard to split , because of inconsistency. The majority of data uses “/t” to split. Yet, there are some sequences with “/t/t”. As a result, many director names are lumped into one big cluster with their movies. As a result, these data sets are lost, and one person, such as Lisa Ann of movie rank 6 of top 100 is inside this huge corrupted data set. As a result, cannot identify the top director for that movie.

Regression

Before doing regression with two features, we test out using one feature first to make sure that we get fair amount of data between those in top directors and not

First feature: Position of director in top director

Most data that we run have position at the 101th bit, which is denoted by number 00000….0001 = 1. There are very few movies which belong to top directors.

Hence, to put some balance for regression for movies directed by top directors,

We sample the movies directed by top directors, by searching valid movie names created by the directors, which are within the threshold constraint (T>10) applied in number 4.

Here are the movie ids (movie id using "movieid" variable) for movies belonging to top directors, and the directors who direct them

```
+   }
+ }
> sampletopdirectormovies
 [1] 444124 444125 444126 444134 164002 358273 213985 216395 169586 344014  77848 481099 455469  68207 396558 374068
> topdirectorsofsample
 [1]  69929  69929  69929  69929 204338 161555  54397  22422 224670  45696  53306 181676 209330 209330 157355 157355
> namesofsampletopdir = director_movies[topdirectorsofsample,1]
> namesofsampletopdir
 [1] "Hackeling, Patrick"     "Hackeling, Patrick"     "Hackeling, Patrick"     "Hackeling, Patrick"     "Smith, Mark Brian"      "Retelas, George"
 [7] "Ganucheau, Chris"       "Dana, Michael (I)"      "Tenenbaum, Daniel"      "Forcella, Matthew (IV)" ""                       "Santana, Miguel (II)"
[13] "Sporns, Charlie"        "Sporns, Charlie"        "Palacios, Cesar"        "Palacios, Cesar"
> |
```

```
> thesamplepositionofdirectors
 [1] 19 19 19 19 30 45 51 56 69 75 83 87 94 94 98 98
```

Above is the sampletopdirectormovies, which are the movie ids (the name of the movie graph nodes), and the name of the top directors directing the sample node movies. The sample movie nodes have its top directors at positions denoted by "thesamplepositionofdirectors"

If the movie has its director in the top 100, then if position is at 3 for example, denote using [0 0 1 0 0 0 ….0] . This number is converted to multiple of 10, which is 10^98 . This is the formula we use

```
regressposnumber = 10^(100-thesamplepositionofdirectors+1)
```

This means that as the position of director is earlier , the more powerful the director is, so a greater position number is assigned.

```
> regressposnumber
 [1] 1e+82 1e+82 1e+82 1e+82 1e+71 1e+56 1e+50 1e+45 1e+32 1e+26 1e+18 1e+14 1e+07 1e+07 1e+03 1e+03
```

```
> ratingofsample
 [1] 9.8 7.2 5.9 7.1 9.8 9.8 9.7 9.7 9.7 9.7
> sampletopdirectormovies
 [1] 444124 444125 444134 164002 213985 216395 344014  77848 455469 374068
> ratingofsample
 [1] 9.8 7.2 5.9 7.1 9.8 9.8 9.7 9.7 9.7 9.7
> topdirectorsofsampple
Error: object 'topdirectorsofsampple' not found
> topdirectorsofsample
 [1]  69929  69929  69929 204338  54397  22422  45696  53306 209330 157355
> namesofsampletopdir = director_movies[topdirectorsofsample,1]
> namesofsampletopdir
 [1] "Hackeling, Patrick"     "Hackeling, Patrick"     "Hackeling, Patrick"     "Smith, Mark Brian"      "Ganucheau, Chris"       "Dana, Michael (I)"
 [7] "Forcella, Matthew (IV)" ""                       "Sporns, Charlie"        "Palacios, Cesar"
```

<u>Explanation for feature on top director position</u>

If we are to regress with only this one feature, then the y = aX+b where y is the ratingofsample, X is 'regressposnumber' which is the position of top director (value between 10^100 to 1). We will do simulation with page rank next

To ensure that we get nodes which belong to top directors, we include those movies under top directors inside our regression, and include 3000 other nodes, we run several nodes from the graph to add some more data points to regress.

```
##doing regression on 1 feature, before adding page rank
fit1<-lm(ratingofsample~regressposnumber)
coef1feat = coefficients(fit1)
res1 = residuals(fit1)
anov1 = anova(fit1)
```

Result of using 1 feature director position in 101 positions

```
> fit1<-lm(ratingofsample~regressposnumber)
> coef1feat = coefficients(fit1)
> res1 = residuals(fit1)
> anov1 = anova(fit1)
> coef1feat
     (Intercept) regressposnumber
    5.896793e+00     1.736540e-82
> anov1
Analysis of Variance Table

Response: ratingofsample
                  Df Sum Sq Mean Sq F value   Pr(>F)
regressposnumber   1   9.01  9.0073  6.8487 0.009066 **
Residuals        687 903.54  1.3152
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Rating_pred = 1.73*10^(-82) * (regressposnumber)+ 5.896

Here, regressposnumber = 10^(100-position+1), the earlier, the better the director achievement

<u>Goodness of fit of one feature</u> (director position)

We compute the goodness of fit by looking at the residual value at the anova table. It shows that the mean square of the residual is 1.3152. The residuals (mean square) using regression with one feature is 1.3152.

<u>Approach of using both page rank and director positions as features</u>

Doing page rank on actor/actress graph with T > 10, I can obtain the page rank of each actor.

On the other hand, previously besides sampling some nodes to find the two feature, I have also obtained some movie nodes whose top director position I have known in order to train the data more accurately , since most of the sampled nodes will yield movies which do not belong to top directors.

To find the actors acting in the movie node, we use the movie_people list which we have created earlier. The movie_people list serves to find the people who have acted in each movie, in terms of people ID. By using people ID, which becomes the name of the nodes in the people graph, we can identify the corresponding page rank values from random walk.

Using these numbers as features, together with regressposnumber (which has been used earlier in one-feature regression) we can do regression to estimate Y based on X1 and X2 to X6. We obtain X2 to X6 by taking only the top 5 values of pageranks of the actors of each movie node. If there are fewer than 5 actors/actresses for this movie node, the remaining variables for the page rank features are assigned zero. For example, if a movie only has 2 actors, then X4-X6 are assigned 0.

Regression using two types of features

Y = a X1 +b X2 + c where X1 is director position with formula the same as before, and X2 is the page rank of the

However, since we will have five features coming from top 5 page ranks of people in each movie, the regression equation becomes

Y = a X1 +b X2 + c X3 +d X4 +e X5 +f X6 +g , where X1 is director position number in terms of one-hot converted into exponent of 10 , as indicated by the same previous formula , and X2 to X6 are top 5 page ranks of each movie's actors (in increasing order)

<u>Result of regression with 2 types of features (page rank and directors) (i.e., 6 features in total)</u>

```
> coef1feat
     (Intercept) regressposnumber              pg1              pg2              pg3              pg4              pg5
    6.045193e+00     1.573400e-82     1.942548e+03     3.046306e+04    -2.746514e+04     3.289681e+03    -5.337015e+03
> anov1 = anova(fit1)
> anov1
Analysis of Variance Table

Response: ratingofsample
                   Df Sum Sq Mean Sq F value   Pr(>F)
regressposnumber    1    8.0  8.0032  5.9827 0.014495 *
pg1                 1    0.0  0.0355  0.0265 0.870590
pg2                 1    0.0  0.0046  0.0034 0.953295
pg3                 1   14.4 14.4403 10.7947 0.001028 **
pg4                 1    0.5  0.4899  0.3662 0.545110
pg5                 1   11.5 11.4664  8.5716 0.003436 **
Residuals        3534 4727.5  1.3377
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Y = 1.573E(-82) X1+ 1.942E3 X2+ 3.0463E4 X3 -2.7465E4 X4 + 3.289E3 X5 -5.337E3 X6 + 6.045

<u>Prediction on the three movies</u>

We first find the directors of the three movies by using their names, in the director_movies file.

```
> rowcol
        row col
[1,] 205496   5
> batmanloc = rowcol
> missioloc = which(c
> missionloc
Error: object 'missic
> missioloc = which(c
> missioloc
     row col
> missioloc = which(c
> missioloc
        row col
[1,] 127707   5
> missionloc = missic
> minionloc = which(c
> minionloc
      row col
[1,] 9259   2
```

```
[1] ""
> director_movies[9258,1]
[1] "Coffin, Pierre"
```

Batman: director row 205496 (Snyder Zack)

Mission: director row 127707 (McQuarrie,Christopher)

Minions: director row 9258 (Coffin Piere)

```
> minionloc[1] = 9258
> directors = c(director_movies[batmanloc[1],1], director_movies[missionloc[1],1], director_movies[minionloc[1],1])
> directors
[1] "Snyder, Zack"            "McQuarrie, Christopher" "Coffin, Pierre"
```

Since the data is corrupted with inconsistent spacing , we cannot get the names of many top directors.

As a result, we cross-check with the top 100 movie ratings that we have obtained earlier through the variable 'moviesorted', which lists out the top 100 movies, to see whether the three directors have directed them . Batman: not inside, Mission: not in , Minions: Not in

Hence, X1 = 1

We will find the X2-X6 of the three movies using movie ID (note pgi = X(i+1) value)

For Batman v Superman

```
> pg1bat
[1] 5.898724e-06
> pg2bat
[1] 7.248838e-06
> pg3bat
[1] 0
> pg4bat
[1] 0
> pg5bat
[1] 0
```

Y = 1.573E(-82) X1+ 1.942E3 X2+ 3.0463E4 X3 -2.7465E4 X4 + 3.289E3 X5 -5.337E3 X6 + 6.045

= 1.573E(-82) ++ 1.942E3(7.25E(-6))+ 6.045 = **6.059**

For Mission Impossible

```
> pg1bat
[1] 4.554574e-06
> pg2bat
[1] 7.037199e-06
> pg3bat
[1] 7.352624e-06
> pg4bat
[1] 0
> pg5bat
[1] 0
```

Y = 1.573E(-82)(1)+ (4.554E-6) 1.942E3 +(7.037E-6)(3.0463E4)+(7.352E-6)(-2.7465E4)+6.045 =**6.0662**

<u>For Minions</u>

```
> pg1bat
[1] 5.898724e-06
> pg2bat
[1] 7.248838e-06
> pg3bat
[1] 0
> pg4bat
[1] 0
> pg5bat
[1] 0
```

Y = 1.573E(-82) (1)+ 1.942E3 (5.8987E-6)+ 3.0463E4 (7.2488E-6) + 6.045 = **6.2772**