EE232E Project02

Silei Ma, Hongyang Li, Nien-Jen Cheng

UID: 604741646 & 304759850 & 804896595

" files" included	
Q1	Problem 1-2.py
Q2	
Q3	Problem 3.R
Q4	Problem 4.py
Q5~Q6	Problem 5.py
	Problem 5-6.R
Q7	7.R
	find_rating.py
Q8	8.R
	find_dir_feature.py
Q9	Problem-9.R
	Problem-9.py

Question 1

We use "**5 movies**" as our threshold. That means only the actors/actresses with acted in at least 5 movies will remain after the filtering.

In this question, we first merge the data of actors and actresses in the ".txt" files and then delete the actors/actresses who act less than 5 movies. Because the movies after the people were spaced by the tabs, we use the following script in Python to withdraw the data more than 4 movies.

```
fields = l.strip().split("\t\t")
if len(fields) > 5:
```

The length of a field is 1+the number of the movies. The "1" is the person who acts on the movies, so we withdraw those lines which length > 5.

Another way to calculate the numbers is to count the "spacings." The numbers of the movies are the numbers of the spacings, done by the tabs. This is more intuitive, because we can just withdraw the movies with actors/actresses more than 4. The idea is shown below.

```
for line in actors_m.readlines():
    if(line.count('\t\t')>4):
```

To speed up the computation, we changed the names of the people into ID numbers. The ID numbers are simply the row numbers in the list. Also, in the process, we remove some special character and the information in the extra parenthesis.

The formant is now

ID# of Person 1	Movie1-1	Movie1-2	Movie1-3	Movie1-4	Movie1-5 …..
ID# of Person 2	Movie2-1	Movie2-2	Movie2-3	Movie2-4	Movie2-5 …..
…					

After the deletion, the number of actors/actresses remain is “244290”.

Question 2

We want to create a directed graph with people as nodes. The weight of an edge for nodes(i →j), is defined below. If person I and j have no common movies, there is no connection between person i and person j.

$$\text{Weight of Nodes(i} \rightarrow \text{j)} = \frac{T\ \ e\ number\ of\ movies\ t\ \ at\ person\ i\ and\ j\ bot\ \ acted}{T\ \ e\ number\ of\ movies\ t\ \ at\ person\ i\ acted}$$

In order to have an elegant algorithm to make the graph, we made a list of movies with their actors/actresses. The data structure of the list is shown below. Furthermore, we also changed the titles of the movies into ID numbers.

ID# of Movie 1	ID# of person1-1	ID# of person1-2	ID# of person1-3	ID# of person1-4 …….
ID# of Movie 2	ID# of person2-1	ID# of person2-2	ID# of person2-3	ID# of person2-4 …….
…				

The list of the movie can be generated by the algorithm shown below.

(1) We find all the movies in “the list of the persons”
(2) List all the movies in the first column in “the list of the movies”
(3) We check every line of persons in “the list of the persons” and find the movie the person acted.
(4) Add the ID# of the person to the “the list of the movies” in a row after those movies the person acted.

Because we changed the titles of the movies into ID numbers, the list was updated into the following form

ID# of Person 1	ID# of Movie1-1	ID# of Movie1-2	ID# of Movie1-3	ID# of Movie1-4 …….
ID# of Person 2	ID# of Movie2-1	ID# of Movie2-2	ID# of Movie2-3	ID# of Movie2-4 …….
…				

Now, it is time to generate the weighted graph list. We can use 3 “for loops” to generate that. (1) The loop of the persons in “the list of persons.” (2) The loop of the movies after the person in “the list of the persons.” (3) The loop of the persons after those movies in “the list of movies.” The algorithm is shown below.

(a) Choose a person in “the list of the persons,” and see what movies the person acted.
(b) Find the movies in “the list of movies.”
(c) In each loop-(2) we record the actors/actress they work with and how many times.
(d) We will get 3 columns with information like this

Persons		Numbers of common movies
ID# of person 1	ID# of person 2 ID# of person 3 ….	N12 N13 ….
ID# of person 2	ID# of person 1 ID# of person 3 ….	N21 N23 ….
ID# of person 3	ID# of person 1 ID# of person 2 ….	N31 N32 ….
……..	….	….

(e) The weight is “the number of common movies” divided by “the number of movies the first person acted.
(f) We save the list in 3 columns as “.txt” shown below so that we can generate the graph in R.

ID# of person 1	ID# of person 2	weight

We used R to open the “txt file” that contains the information and then generated the graph. Due to the data size of the graph is too large, we can not print the graph here. The code is included in our package, one can recover the graph by running our code.

The directed graph.

Number of edges within the graph = 57846322
Number of vertices = 244290

Question 3

In this section, we are going to conduct the page rank of the actors/actress via the algorithm. We will list the top 10 of them. Also, we will also list the page rank of them based on our experience.

Top 10 via the algorithm				
	Actors/Actresses #	Actors/Actress names	Page Rank	# of movies acted
1	180967	Flowers, Bess	0.0001576882	828
2	137186	Tatasciore, Fred	0.0001399245	355
3	13716	Blum, Steve (IX)	0.0001343229	373
4	57602	Harris, Sam (II)	0.0001331094	600
5	66949	Jeremy, Ron	0.0001228220	637
6	94326	Miller, Harold (I)	0.0001163193	561
7	83800	Lowenthal, Yuri	0.0001097413	318
8	37713	'Downes, Robin Atkin	0.0001047281	267
9	109530	Phelps, Lee (I)	0.0001009891	647
10	124110	Sayre, Jeffrey	0.0001005030	430

Top 10 in our opinion				
	Actors/Actresses #	Actors/Actress names	Page Rank	# of movies acted
1	26672	Clooney, George	3.247805e-05	67
2	41913	Evans, Chris (V)	1.908539e-05	36
3	110499	Pitt, Brad	3.367534e-05	71
4	130682	Smith, Will (I)	2.587971e-05	49
5	67506	Johnson, Dwayne	3.336329e-05	78
6	59284	Hemsworth, Chris	1.492367e-05	21
7	231956	Stone, Emma (III)	1.700542e-05	26
8	243795	Zhang, Ziyi	8.542322e-06	32
9	175141	Dench, Judi	2.826469e-05	73
10	200117	Lawrence, Jennifer (III)	1.364611e-05	28

We have seen that some famous actors don't have front page ranks. The reason is that the fame is not totally related to the page rank. The page rank means the time people search for. However, for example, every person knows George Washington, but they won't google "George Washington" all the times. In contrast, not everyone knows the President of Philippines, Mr. Rodrigo Duterte, but people find information about him to know better. Just like in the graph, the new stars may not as famous as the Oscar awarded actors, but they have better ranks. In another words, we may say that the better the ranks, the faster their reputation increase. They may not that famous at this time, but if we recover the graph few years later, they may turn into super stars.

Another reason is that those people who were found in top page ranks acted in hundreds of movies while the real famous ones with relatively lower page ranks acted only in less than a hundred movies.

Usually, the “passersby” actors/actress appear in the movies only for a few minutes or even only for few seconds. Although they acted in so many movies, but they are not important at all. Therefore, no one will remember them.

We can infer that, if we want this data base to be more reliable on prediction the reputation. We must only include the movies they acted in, but also include **the DURATIONS in those movies**. Furthermore, to make the # of movies they acted more important, we should include a parameter that the movies they acted per year in their movie careers. Or to be more specific, the **TOTAL DURATIONS PER YEAR** in their movie careers.

As for the relation between the two tables (The algorism one and the one with our opinion), we did not see any pairing between the two. Based on the inference written above, it was not surprised that well-known actors/actresses did not show in the first table. After all, the database did not provide the information of the **DURATION** they exist in those movies.

Question 4

We use “**5 actors/actresses**“ as our threshold. That means only the movies have at least 5 actors/actresses will remain after the filtering.

We already have the list “Movie ID’s to their actors/actresses” in Question 2. Now, we are going to filter the data that only the movies with more than actors/actresses will remain. The separations were done by the tabs, so just like the method we used in question 1, we can recover a list that the movies can remain if only if the length of the field is greater than 5. Note that, the length of the field is the number of persons plus 1, the movie.

```
fields = l.strip().split("\t\t")
if len(fields) > 5:
```

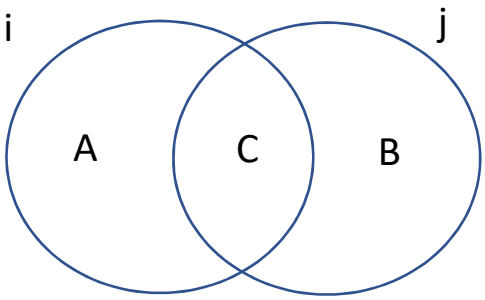
The new list was saved in the form below.

ID# of Movie 1	ID# of person1-1	ID# of person1-2	ID# of person1-3	ID# of person1-4 …….
ID# of Movie 2	ID# of person2-1	ID# of person2-2	ID# of person2-3	ID# of person2-4 …….
…				

Now, we want to generate a graph that the nodes are the movies and the edges represent the common actors/actresses. Unlike the persons’ graph is question 2, which is a directed graph, the graph that we are generating now is an undirected graph. The weight of an edge is defined below, which is the jaccard index.

$$\text{Weight of Nodes(i , j)} = \frac{T\ \ e\ number\ of\ actors/actresses\ t\ \ at\ movie\ i\ and\ j\ bot\ \ \ \ ave}{T\ \ e\ total\ number\ of\ actors/actresses\ in\ t\ \ e\ 2\ movies}$$

We can show this by a Venn diagram shown below. Node i and j are two sets, and A, B, C are the numbers of actors/actresses in each area.



$$\text{Weight of Nodes(i , j)} = \frac{C}{A+B+C}$$

In our code, to have a neat algorithm, we write them as the form below.

$$\text{Weight of Nodes(i , j)} = \frac{[Persons\ in\ Node\ i] \cap [Persons\ in\ Node\ j]}{[Persons\ in\ Node\ i] + [Persons\ in\ Node\ j] \quad ([Persons\ in\ Node\ i] \cap [Persons\ in\ Node\ j])}$$

We again save them into a list of ".txt" in the form below.

ID# of movie 1	ID# of movie 2	weight

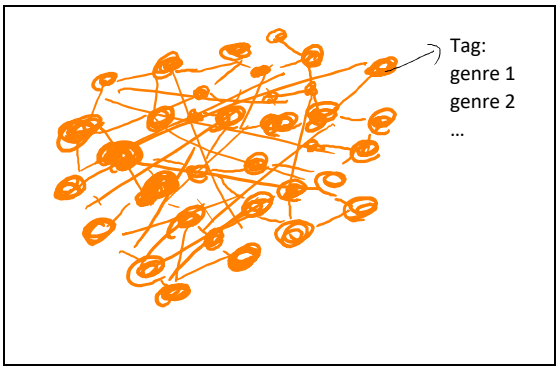
We used R to open the "txt file" that contains the information and then generated the graph. Again, the size of the data is too large that we can not print them here. The code is included in our package, one can recover the graph by running our code.

The undirected graph.

Number of edges within the graph = 70640020
Number of vertices = 246378

Question 5

So far, we already recovered the graph in question 4. It is easy to construct the communities via the method in the previous home works. The Fast Greedy Newman algorithm was utilized this time. In our experience, the "Fast Greedy approach" usually generate good communities that only the close-related nodes will stay in the same community. Due to the data was too big, our computational devices crashed every time when plotting the communities. However, we proposed a quick draft below to show the idea.



We can import the genre information into the graph in question 4 via the code below.

```
V(movie_graph)$genre = gen_mov[as.numeric(V(movie_graph)$name)]
```

Then, we can withdraw the genres which have more percentage more than 20%.

```
for(genre in genre_type){
  if(length(which(comm_genre==genre))>=length(comm_genre)*0.2 && genre!="Null"){
    comm_tag<-c(comm_tag,genre)
  }
 }
```

There were **748 communities** found in the graph. We list those communities which have movies greater than 10. 50 of the communities have movies greater than 10. The detail information of the 50 communities are shown in the table below. N/A means that all the genres tags were less than 20%.

Commty #	Genres	# of movies		Commty #	Genres	# of movies
1	Drama	61909		38	Short	11
2	Short	145094		39	Short Drama	12
3	Drama	8531		80	Short Thriller	27
4	Adult	5222		125	Drama	35
5	Drama Short	7504		181	Drama	219
6	Drama	16836		182	Sport	19
7	Drama Comedy	3178		217	Drama Short	37
8	Drama	8503		284	Short	17
9	Drama	6838		305	Comedy Thriller	21
10	Drama	10744		309	Short	12
11	Short	40083		335	Short Thriller	28
12	Drama	10015		353	Short	15
13	Drama	12740		401	Short	19
14	Short	2057		403	Short	24
15	Short	29		409	Short	12

16	Short Drama	3517		421	Short	14
17	-N/A-	1275		422	Short Horror	14
18	Drama	1993		561	Short	12
19	Drama Short	2570		601	Short Thriller	16
21	Drama Romance	4641		643	Short	11
22	Drama	1284		655	Short	29
25	Sport	101		703	Short	13
29	Drama	1779		713	Thriller	15
32	Short Thriller	60		732	Documentary	13
33	Action Short	13		743	Thriller Short	15

From the list, we can see that the genres, short and drama are the most common ones. We know that the movies are related via the actors/actresses. We can infer that one actor/actress tend to act in movies of similar genres, so in one community, we can see similar type of actors/actresses. Therefore, the tags are meaningful. One thing has to be mentioned is that most of the tags only appear in the percentage lower than 30%, so a tag cannot represent the whole community.

The connections between the movies were made out of the common actors/actress, but again, just like we mention in the question 3, the data did not provide the TOTAL DURATION of the actors/actresses. Therefore, the tags were not that accurate. We should add the actors'/actresses' weights, which are the durations of the existing time in the movies, so that the tags can reveal "human-intuitive results".

Question 6

In this question set, 3 nodes will be added into the graph of movies. The 3 movies are.

Batman v Superman: Dawn of Justice (2016)
Mission: Impossible - Rogue Nation (2015)
Minions (2015)

First, we have to grant them movie ID's and replace the names of actors/actresses by person ID's. If there are new actors/actresses, we have to update the list of actors/actresses and grant them with person ID's as well. However, we found that all the 3 movies have already existed in the library, so we can just call the 3 movies from the library. The ID of the 3 movies in our library are, "894354," "779751," and "763763." We can call them by the commend shown below.

```
movieDict[movNO[894354]]
movieDict[movNO[779751]]
movieDict[movNO[763763]]
```

The detail of the three movies are shown below.

Movie name	Movie ID	Actors'/Actresses' ID's in the movie
Batman v Superman: Dawn of Justice (2016)	894354	4809, 10924, 15056, 29462, 31949, 36722, 37662, 46327, 55330, 96488, 118333, 135508, 185859, 189245, 199152, 199611, 224832, 233823, 247373, 259355, 260729, 279201, 300313, 311405, 318114, 333259, 351868, 355639, 381274, 382876, 411068, 416759, 422044, 427032, 441663, 461567, 474483, 479988, 534563, 549594, 556949, 610602, 610612, 611528, 655178, 664481, 679816, 707454, 724503, 736627, 763229, 771174, 793581, 814584, 817072, 817437, 822401, 846011, 849099, 899831, 920967, 922688, 930135, 931482, 1036873, 1062409, 1090204, 1099591, 1101657, 1115040, 1130253, 1147936, 1151854, 1153322, 1156141, 1165530, 1242203, 1242638, 1253260, 1258552, 1272877, 1295188, 1297583, 1300098, 1306327, 1327689, 1344180, 1361732, 1364526, 1378557, 1389141, 1393404, 1398572, 1419959, 1450235, 1465940, 1474086, 1477696, 1484601, 1538637, 1556871, 1578698, 1579652, 1632663, 1641212, 1669893, 1681340, 1691377, 1717117, 1717122, 1744672, 1751010, 1759452, 1809587, 1848595, 1876910, 1921156, 1947924, 1952344, 1957069, 2002778, 2027216, 2047941, 2088572, 2091524, 2119956, 2156367, 2172340, 2215654, 2264207, 2308006, 2311381, 2345585, 2360934, 2390550, 2391481, 2400017, 2401573, 2429767, 2476191, 2504056, 2509171, 2525637, 2526586, 2539096, 2541340, 2557674, 2562650, 2627199, 2634961, 2644637, 2665520, 2674310, 2701681, 2726188, 2735425, 2754122, 2774666, 2814484, 2824139, 2840698, 2879867, 2933063, 2949285, 2958151, 3009479, 3046272, 3048197, 3096927, 3105336, 3106240, 3139051, 3143780, 3167768, 3283983, 3299235, 3322357, 3325417
Mission: Impossible - Rogue Nation (2015)	779751	8452, 25145, 37788, 59877, 61640, 84990, 85621, 89133, 91395, 107831, 114013, 116769, 163815, 174713, 221069, 223436, 261143, 301824, 367868, 376411, 400139, 411302, 414016, 417395, 428514, 507664, 540762, 590286, 602614, 624160, 720440, 732526, 749456, 782637, 785880, 792684, 797576, 805695, 812324, 868514, 868956, 870296, 874446, 878689, 893218, 913189, 929976, 966874, 998429, 1005472, 1095317, 1112457, 1115925, 1144589, 1150240, 1177166, 1197826, 1220091, 1247837, 1251904, 1258698, 1276369, 1372703, 1389695, 1399222, 1436779, 1496637, 1499009, 1520188, 1537188, 1550303, 1577581, 1579019, 1584140, 1611906, 1617906, 1641568, 1686966, 1725824, 1736606, 1768748, 1847904, 1854273, 1904694, 1916965, 1996543, 2029613, 2065525, 2066354, 2077038, 2150749, 2160283, 2168475, 2208555, 2228199, 2245578, 2257970, 2261688, 2275783, 2277524, 2305748, 2378373, 2380102, 2493265, 2494362, 2514074, 2567742, 2579252, 2600547, 2600706, 2635990, 2666694, 2667044, 2773146, 2783759, 2825441, 2832116, 2923516, 2933522, 2939180, 2947633, 2960616, 2985622, 3061087, 3136363, 3186813, 3216835, 3264454, 3308355, 3312051, 3341556, 3343674
Minions (2015)	763763	140125, 160859, 188751, 221564, 368999, 385637, 423695, 471106, 586024, 781422, 793539, 987275, 1204543, 1577732, 1611097, 1621053, 1664005, 1706097, 2308121, 2664947, 2809466, 2860733, 2891585, 2935347, 3109850

The top 5 nearest neighbors of the 3 new nodes are shown below.

Batman v Superman: Dawn of Justice (2016) in [Community 2]		
Movie ID	Movie Names	Community
991910	Get the Hell Out of Hamtown (1999)	2
399163	Eloise (2015)	2
914813	Love in Dead Places (2013)	2
1003512	Behind the Scenes of Horrorcore Hotel (2014)	2
301856	Into the Storm (2014)	2

Mission: Impossible - Rogue Nation (2015) in [Community 2]		
Movie ID	Movie Names	Community
818741	Now You See Me: The Second Act (2016)	2
1008093	Dusha shpiona (2015)	2
672803	Fan (2015)	2
765616	The Program (2015/II)	2
429736	Breaking the Bank (2014)	2

Minions (2015) in [Community 2]		
Movie ID	Movie Names	Community
200580	Cars (2006)	2
228519	The Lorax (2012)	2
376170	Despicable Me 2 (2013)	2

565673	Ice Age: The Meltdown (2006)	2
533166	Monsters University (2013)	2

The three movies belong to the same community, which is community 2. The tag of the community is "short".

In question 5, we have mentioned that the percentages of most of the tags were below 30%. Hence, it is not surprised that the genres of the movies are not the same as those of the communities.

We were very happy about the result we found. The neighbors of one movie of the 3 are the very related the one movie. Especially the Minions (2015), which is a cartoon, we were very surprised that all the neighbors we found were all cartoons, too. We think the reason may be that the actors/actresses of cartoons usually are vocal actors/actresses, or we call them "senyo" in Japanese. Therefore, the cartoons are quite correlated.

Question 7

We are now predicting the rating of the 3 new movies listed in question 6 based on other movies in the graph. Because similar movies tend to have same rating. We will predict their rating via the rating in their communities and their neighbors. There are several methods. In the following, we denote the rating of the neighbors as Rn(x) and the rating of the community as Rc(y), where x ∈ neighbors and y ∈ community.

Method 1

We can use only the neighbors to derive the result. The Final rating R_f will be:

$R_f = \sum_{x=1}^{X} \mathrm{Rn(x)}/\mathrm{X}$

X is the numbers of the set.

Method 2

Also, we can use only the community information to derive the result. The Final rating R_f will be:

$R_f = \sum_{y=1}^{Y} \mathrm{Rc(y)}/\mathrm{Y}$

Y is the numbers of the set.

Method 3

We first find the average of Rn(x) and Rc(y). We denote them as $\sum_{x=1}^{X} \mathrm{Rn(x)}/\mathrm{X}$ and $\sum_{y=1}^{Y} \mathrm{Rc(y)}/\mathrm{Y}$. X and Y are the numbers of each sets.

Second, we make average of the two averages. Hence the Final rating R_f will be:

$$R_f = \frac{\sum_{x=1}^{X} \mathrm{Rn(x)}/X + \sum_{y=1}^{Y} \mathrm{Rc(y)}/Y}{2}$$

Method 4

Unlike finding their averages respectively, we sum all the Rn(x) and Rc(y) and then divide them by their number.

$$R_f = \frac{\sum_{x=1}^{X} \mathrm{Rn(x)} + \sum_{y=1}^{Y} \mathrm{Rc(y)}}{X+Y}$$

Method 5

In the 3rd method, we include the weight of the edges in each neighbors or community members. The averages in method 1 will be modified to $\sum_{x=1}^{X} \mathrm{Rn(x)} \quad \mathrm{Wn(x)}/X$ and $\sum_{y=1}^{Y} \mathrm{Rc(y)} \quad \mathrm{Wc(y)}/Y$. The rating is now shown below.

$$R_f = \frac{\sum_{x=1}^{X} \mathrm{Rn(x)} \;\; \mathrm{Wn(x)}/X + \sum_{y=1}^{Y} \mathrm{Rc(y)} \;\; \mathrm{Wc(y)}/Y}{2}$$

Method 6

We again include the weights, but this time, we include them in method 2.

Now,

$$Rf = \frac{\sum_{x=1}^{X} \mathrm{Rn(x)} \;\; \mathrm{Wn(x)}/X + \sum_{y=1}^{Y} \mathrm{Rc(y)} \;\; \mathrm{Wc(y)}/Y}{X+Y}$$

The result of the 4 methods are shown in the table below.

Movie name	Batman v Superman: Dawn of Justice (2016)	Mission Impossible - Rogue Nation (2015)	Minions (2015)
Actual on IMDb	7.1	7.5	6.4
Method 1	6.110256	5.99537	6.396138
Method 2	6.074895	6.129327	6.195559
Method 3	6.275912	6.211204	6.512196
Method 4	6.191796	6.189184	6.19809
Method 5	5.383655	5.26059	6.360177
Method 6	6.158682	6.162731	6.19361

Question 8

In this question, we want to predict the rating of a movie from the following features.

Feature #	Feature description	Values
Feature 1	5 page rank values of the top 5 actors/actresses in the movie	5 floating point values
Feature 2	If the director is who has produced a top-100 movie.	0 or 1

We can make the features into a 6-dimensional vector, and then feed them into a classifier. We first train the model via the known data, then we can use the model to predict the rating of the 3 movies.

In statistics, linear regression is an approach for modeling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables) denoted X. The case of one explanatory variable is called simple linear regression. In this question, we simply use the built-in function in R to build a regression model. We first extract the features from the data from previous questions. And then apply these features and the result to the function. Therefore, we get the model. And at last we can get the predicted rating of the 3 movies mentioned in this question.

We read the page rank from a ".txt," and read the ".txt" for feature 2. The code is shown below.

```
page_rank_movies = fread("page_rank_movies.txt", header = FALSE)
movie_graph$p_r = page_rank_movies

feature_director = fread("feature_director.txt", header = FALSE)
movie_graph$director = feature_director
```

Then, the below code feed the features into the classifier.

```
model.reg = lm(Ratings ~ ., data = model.data)
```

The result is shown in the below table

Movie	Predicted Rating
Batman v Superman: Dawn of Justice (2016)	6.040144
Mission: Impossible - Rogue Nation (2015)	5.870426
Minions (2015)	6.044278

Question 9

First, we need to create a rating list of the actors/actresses. The rating of a person is the average rating of all the listed movies one acted. The formula is shown below. We denote them as Rp(p), where p stands for the index of a person. Total movie in the list is denoted as L. The Rating of one movie is denoted as M(l).

$$Rp(p) = \frac{\sum_{l=1}^{L} M(l)}{L}$$

Then, we can predict the rating of a movie based on the rating of the actors/actresses in that movie. Note that the influence of each persons are different, so we need to split the persons into classes. In our approach, we split the persons into 3 classes, Star, Normal and Rookie. Now, we have to design the weights and rating intervals of each classes. We have tested several parameters in this section, and the parameters shown in the table below gave us quite good result.

Type	Rating interval	Weight
Star 	$7 \leq Rating \leq 10$	10
Normal 	$5.5 \leq Rating < 7$	3
Rookie 	$Rating < 5.5$	1

Then, we can generate the graph of the people rating list in R. The information of the graph is shown below.

Number of edges within the graph = 3429608
Number of vertices = 487734

Now, it is time to find the rating of a movie from its actors/actresses.

The rating of the movie Rm can be derived from the formula shown below. The total number in the list of the movie is P.

$$Rm = \sum_{p=1}^{P} W(p) \quad Rp(p) = \frac{Total\ rating\ score\ based\ on\ weighting}{(\#\ \text{of Star})\ 10 + (\#\ \text{of Normal})\ 3 + (\#\ \text{of Rookie})\ 1}$$

We can recover the result via the code shown below.

```
for i in range(1,len(movieDict[movNO[movID]])):
    actID = movieDict[movNO[movID]][i]
    actscore = ScoAct[actNO[actID]]
    if actscore != 0:
        if actscore > 7:
            score = score + 10*actscore
            star = star + 1
        elif actscore <= 7 and actscore > 5.5:
            score = score + 3*actscore
            normal = normal + 1
        else:
            score = score + actscore
            rookie = rookie + 1
av_score = score/(rookie + 3*normal + 10*star)
```

The Predicted Rating of the 3 movies are shown in the below table.

Movie	Predicted Rating
Batman v Superman: Dawn of Justice (2016)	7.133434
Mission: Impossible - Rogue Nation (2015)	7.059891
Minions (2015)	7.105581