Meropi Argyropoulou BAFT 1722

Aglaia Tsopelakou

BAFT 1716

Data Mining Techniques Assignment 1

Contents

[1. Process the original dataset 2](#_Toc515914898)

[Create Undirected Graph 2](#_Toc515914899)

[Creation of Friends Function 2](#_Toc515914900)

[2. Recommending friends using Common neighbors (friend-of-friend (FoF) method) 3](#_Toc515914901)

[3. Recommending friends using Jaccard coefficient 5](#_Toc515914902)

[4. Recommending friends using Adamic and Adar function 7](#_Toc515914903)

[5. Evaluation of the recommendation system 9](#_Toc515914904)

[First Step: 9](#_Toc515914905)

[Second Step: 11](#_Toc515914906)

[6. Assignment Bonus 15](#_Toc515914907)

[Alternative Evaluation Method: 15](#_Toc515914908)

[Additional Scoring Function: 17](#_Toc515914909)

Code Tables

[Code Table 1 2](#_Toc515914875)

[Code Table 2 3](#_Toc515914876)

[Code Table 3 4](#_Toc515914877)

[Code Table 4 5](#_Toc515914878)

[Code Table 5 7](#_Toc515914879)

[Code Table 6 10](#_Toc515914880)

[Code Table 7 13](#_Toc515914881)

[Code Table 8 16](#_Toc515914882)

[Code Table 9 20](#_Toc515914883)

Results Tables

[Results Table 1 4](#_Toc515914757)

[Results Table 2 6](#_Toc515914758)

[Results Table 3 8](#_Toc515914759)

[Results Table 4 11](#_Toc515914760)

[Results Table 5 21](#_Toc515914761)

Figures

[Figure 1 11](#_Toc515914775)

[Figure 2 13](#_Toc515914776)

[Figure 3 14](#_Toc515914777)

[Figure 4 17](#_Toc515914778)

[Figure 5 21](#_Toc515914779)

**Recommending Friends**

To solve the questions of this assignment R programming language was used and R studio.

# Process the original dataset

## Create Undirected Graph

In order to transform the graph to undirected, we create a new data frame *“c”* (line 5) in R where the 1st column has as nodes those of the 2nd column of the initial dataset and the 2nd column has as nodes those of the 1st column of the initial dataset. In this way we create the opposite direction of the nodes from the initial dataset. In line (8) we merge the initial dataset with the new one we created. We then ensure that there are no duplicate pairs of nodes by using the *“unique”* command.

**R code used:**

#load the data to r-studio

1. facebook**<-**read.table**(**"facebook\_combined.txt", sep**=**" "**)**
2. nrow**(**facebook**)** #88234 rows
3. a**<-**facebook**[**,2**]** #2nd column of facebook dataframe
4. b**<-**facebook**[**,1**]** #1st column of facebook dataframe
5. c**<**-as.data.frame**(**cbind**(**a, b**))** #combine the two columns in one new dataframe to create the opposite direction

#change the names

1. names**(**facebook**)**
2. names**(**c**)<-**c**(**"V1", "V2"**)**

#"merge" the initial dataset facebook with the new one that has the

opposite directions

1. c**<-**rbind**(**facebook,c**)**
2. nrow**(**c**)**#176468

#ensure the uniqueness of all pairs.

1. nrow**(**unique**(**c**))** #176468

Code Table 1

## Creation of Friends Function

We created the following function because its output is used by all methods, Fof, Jaccard and Adamic. This function takes as input a target node and the undirected graph *“c”* created from the previous step and returns a list that contains as first element the friends of the target node*(“target”)* and a list *(“lista”)* that contains the friends of those nodes that are **not** friends with the target node.

**R code used:**

1. friends**<-function(**node, c**){**
2. #friends of node

target**<-**c**[**c**$**V1**==**node,2**]**

1. #all users(nodes)

unique\_users**<-**unique**(**c**$**V1**)**

1. #Users that are not friends of #node

notfriends**<-**setdiff**(**unique\_users,target**)**

1. #Exclude the target node

notfriends**<-**notfriends**[!**notfriends%in%c**(**node**)]**

1. library**(**"rlist"**)**
2. #List of friends of not friends of target (target is not included as friend)
3. lista**<-**list**()**
4. **for** **(**i **in** notfriends**){**
5. lista**<-**list.append**(**lista, c**[**c**$**V1**==**i,**])}**
6. return **(**list**(**target, lista**))}**

**Calling with specific node and the undirected dataset:**

1. a107**<-**friends**(**107, c**)**
2. a1126**<-**friends**(**1126, c**)**
3. a14**<-**friends**(**14, c**)**
4. a35**<-**friends**(**35, c**)**
5. a7**<-**friends**(**7, facebook\_sample**)**

Code Table 2

# Recommending friends using Common neighbors (friend-of-friend (FoF) method)

We created the following function to implement the FoF method. In detail, the logic behind follows:

1. Finds those nodes that are not friends of the target.
2. For these nodes it calculates the number of common friendships with the target.
3. It sorts the results for the target node and presents the first 10.

**R code used:**

fof\_fun**<-function(**target, lista**){**

#saves the common friends of those nodes that are not friends of the target

**for** **(**i **in** 1**:**length**(**lista**)){**lista**[[**i**]]<-**list**(**lista**[[**i**]]**,intersect**(**target, lista**[[**i**]]$**V2**))}**

#saves the number of common friends of target with the friends of those nodes that are not friends of the target

**for** **(**i **in** 1**:**length**(**lista**)){**lista**[[**i**]]<-**list**(**lista**[[**i**]]**,length**(**lista**[[**i**]][[**2**]]))}**

#orders the results based on the largest number of mutual friends and keeps the first 11

listb**<-**head**(**lista**[**order**(**sapply**(**lista, **function(**x**)** x**[[**2**]]**, simplify**=TRUE)**, decreasing**=TRUE)]**,11**)**

#formats the results

cur**<-NULL**

**for** **(**i **in** 1**:**length**(**listb**))**

**{**cur**<-**rbind**(**cur,data.frame**(**unique**(**listb**[[**i**]][[**1**]][[**1**]]$**V1**)**,listb**[[**i**]][[**2**]]))}**

names**(**cur**)<-**c**(**"NodeID","Score"**)**

#In the case of ties in friendship score you should output the node with the smallest nodeID.

**if** **(**cur**[**nrow**(**cur**)**,2**]==**cur**[**nrow**(**cur**)-**1,2**]){**

**if** **(**cur**[**nrow**(**cur**)**,1**]==**max**(**cur**[**nrow**(**cur**)**,1**]**, cur**[**nrow**(**cur**)-**1,1**])){**

cur**<-**cur**[-**nrow**(**cur**)**,**]}**

**else{**

cur**<-**cur**[-(**nrow**(**cur**)-**1**)**,**]}**

**}**

**else** **{**cur**<-**cur**[-**nrow**(**cur**)**,**]}**

return **(**cur**)**

**}**

**Calling with parameters friends list of target and the friends of the non-friends of target:**

fof\_fun**(**a107**[[**1**]]**, a107**[[**2**]])**

fof\_fun**(**a1126**[[**1**]]**, a1126**[[**2**]])**

fof\_fun**(**a14**[[**1**]]**, a14**[[**2**]])**

fof\_fun**(**a35**[[**1**]]**, a35**[[**2**]])**

Code Table 3

**Results:**

The first 10 recommendations and their corresponding score after running the FoF method are listed below per the target node ID requested.

|  |  |
| --- | --- |
| nodeID | Recommendation || Score |
| 107 | NodeID ||Score  513 19  400 18  559 18  373 17  492 17  500 17  378 16  436 16  431 15  514 15 |
| 1126 | NodeID Score  916 113  1238 103  1750 99  1230 98  1004 94  1791 94  1530 90  1172 89  1570 85  1597 84 |
| 14 | NodeID Score  2 9  17 9  140 9  111 8  137 7  162 7  19 6  333 6  44 5  243 4 |
| 35 | NodeID Score  46 2  68 2  99 2  131 2  175 2  177 2  225 2  227 2  278 2  321 2 |

Results Table 1

# Recommending friends using Jaccard coefficient

We created the following function to implement the Jaccard method. In detail, the logic behind follows:

1. Finds those nodes that are not friends of the target.
2. For these nodes it calculates the number of common friendships with the target.
3. Calculates the number of nodes of the union set of the friends of target and the friends of not friends.
4. Computes the division of 2./3. and save it as a score.
5. Sorts the results and presents the first 10.

**R code used:**

jaccard\_fun**<-function(**target, lista**){**

#saves the common friends of those nodes that are not friends of the target

**for** **(**i **in** 1**:**length**(**lista**)){**lista**[[**i**]]<-**list**(**lista**[[**i**]]**,intersect**(**target, lista**[[**i**]]$**V2**))}**

#The change is here for Jaccard!!! We calculate the division of the length of the common friends of target and the friends of those nodes that are not friends of target with the length of the union of the not friends of target.

**for** **(**i **in** 1**:**length**(**lista**)){**

lista**[[**i**]][[**1**]]<-**list**(**lista**[[**i**]][[**1**]]**,length**(**lista**[[**i**]][[**2**]])/**length**(**union**(**lista**[[**i**]][[**1**]]$**V2,target**)))}**

#The change is also here for Jaccard!!! Orders the results based on the largest number of mutual friends and keeps the first 11 results

listb**<-**head**(**lista**[**order**(**sapply**(**lista, **function(**x**)** x**[[**1**]][[**2**]]**, simplify**=TRUE)**, decreasing**=TRUE)]**,11**)**

#The change is also here for Jaccard!!!-formats the results

cur**<-NULL**

**for** **(**i **in** 1**:**length**(**listb**))**

**{**cur**<-**rbind**(**cur,data.frame**(**unique**(**listb**[[**i**]][[**1**]][[**1**]]$**V1**)**,listb**[[**i**]][[**1**]][[**2**]]))}**

names**(**cur**)<-**c**(**"NodeID","Score"**)**

**if** **(**cur**[**nrow**(**cur**)**,2**]==**cur**[**nrow**(**cur**)-**1,2**]){**

**if** **(**cur**[**nrow**(**cur**)**,1**]==**max**(**cur**[**nrow**(**cur**)**,1**]**, cur**[**nrow**(**cur**)-**1,1**])){**

cur**<-**cur**[-**nrow**(**cur**)**,**]}**

**else{**

cur**<-**cur**[-(**nrow**(**cur**)-**1**)**,**]}}**

**else** **{**

cur**<-**cur**[-**nrow**(**cur**)**,**]}**

return **(**cur**)}**

**Calling with parameters friends list of target and the friends of the non-friends of target:**

jaccard\_fun**(**a107**[[**1**]]**, a107**[[**2**]])**

jaccard\_fun**(**a1126**[[**1**]]**, a1126**[[**2**]])**

jaccard\_fun**(**a14**[[**1**]]**, a14**[[**2**]])**

jaccard\_fun**(**a35**[[**1**]]**, a35**[[**2**]])**

Code Table 4

**Results:**

The first 10 recommendations and their corresponding score after running the Jaccard method are listed below per the target node ID requested.

|  |  |
| --- | --- |
| nodeID | Recommendation || Score |
| 107 | NodeID Score  513 0.01704036  400 0.01636364  559 0.01621622  492 0.01555352  500 0.01530153  373 0.01515152  436 0.01469238  378 0.01466544  515 0.01374885  514 0.01371115 |
| 1126 | NodeID Score  916 0.4870690  1750 0.4400000  1230 0.4355556  1530 0.4225352  1004 0.4196429  1238 0.4170040  1172 0.4009009  1791 0.4000000  1789 0.3744292  1597 0.3716814 |
| 14 | NodeID Score  2 0.5625000  140 0.5294118  7 0.4736842  162 0.4375000  111 0.3809524  333 0.3529412  44 0.3125000  137 0.2916667  19 0.2400000  243 0.2105263 |
| 35 | NodeID Score  321 0.6666667  11 0.5000000  12 0.5000000  15 0.5000000  18 0.5000000  37 0.5000000  43 0.5000000  74 0.5000000  114 0.5000000  209 0.5000000 |

Results Table 2

# Recommending friends using Adamic and Adar function

We created the following function to implement the Adamic-Adar method. In detail, the logic behind follows:

1. Finds those nodes that are not friends of the target.
2. For each one of these nodes find their friends and take the intersection set of these friends with the friends of the target.
3. For every node (common friend) that appears in the resulting intersection set, find the number of friends it has(N) and calculate the 1/log(N). Iterate through the intersection set and do the summation of all 1/log(N). This is considered as the score.
4. Sort the results and present the first 10.

**R code used:**

ada**<-function(**c, target, lista**){**

#saves the common friends of those nodes that are not friends of the target

**for** **(**i **in** 1**:**length**(**lista**)){**

lista**[[**i**]]<-**list.append**(**lista**[[**i**]]**, list**(**intersect**(**target, lista**[[**i**]]$**V2**)))}**

listq4**<-**list**()**

#For every non friend of target

**for** **(**i **in** 1**:**length**(**lista**)){**

sum**<-**0

#For every common friend that the target has with the non friend, calculate the number of friends that it has, take its 1/logarithm and save it.

**for** **(**j **in** 1**:**length**(**unlist**(**lista**[[**i**]][[**3**]]))){**

sum**<-**sum**+**1**/**log10**(**length**(**c**[**c**$**V1**==**lista**[[**i**]][[**3**]][[**1**]][**j**]**,2**]))}**

listq4**<-**list.append**(**listq4, list**(**unique**(**lista**[[**i**]][[**1**]])**, sum**))}**

#order the results and keep the first 11

listb**<-**head**(**listq4**[**order**(**sapply**(**listq4, **function(**x**)** x**[[**2**]]**, simplify**=TRUE)**, decreasing**=TRUE)]**,11**)**

#formats the results

df**<-**data.frame**()**

**for** **(**i **in** 1**:**length**(**listb**)){**

df**<-**rbind**(**df, t**(**data.frame**(**unlist**(**listb**[[**i**]]))))}**

rownames**(**df**)<-NULL**

names**(**df**)<-**c**(**"NodeID","Score"**)**

**if** **(**df**[**nrow**(**df**)**,2**]==**df**[**nrow**(**df**)-**1,2**]){**

**if** **(**df**[**nrow**(**df**)**,1**]==**max**(**df**[**nrow**(**df**)**,1**]**, df**[**nrow**(**df**)-**1,1**])){**

df**<-**df**[-**nrow**(**df**)**,**]}**

**else{**

df**<-**df**[-(**nrow**(**df**)-**1**)**,**]}**

**}**

**else** **{**

df**<-**df**[-**nrow**(**df**)**,**]}**

return**(**df**)**

**}**

**Calling with parameters friends list of target and the friends of the non-friends of target:**

ada**(**c, a107**[[**1**]]**, a107**[[**2**]])**

ada**(**c, a1126**[[**1**]]**, a1126**[[**2**]])**

ada**(**c, a14**[[**1**]]**, a14**[[**2**]])**

ada**(**c, a35**[[**1**]]**, a35**[[**2**]])**

Code Table 5

**Results:**

The first 10 recommendations and their corresponding score after running the Adamic method are listed below per the target node ID requested.

|  |  |
| --- | --- |
| nodeID | Recommendation || Score |
| 107 | NodeID Score  513 9.724953  400 9.307010  559 8.818876  500 8.528682  492 8.373501  373 8.368363  378 7.852810  436 7.813676  524 7.441609  514 7.437235 |
| 1126 | NodeID Score  916 53.55625  1238 48.75375  1750 46.49691  1230 45.81190  1004 44.04181  1791 43.95921  1530 41.80850  1172 41.57966  1570 40.00642  1597 39.14693 |
| 14 | NodeID Score  2 6.843324  17 6.736628  140 6.736628  111 5.964252  162 5.214517  137 5.073642  333 4.598281  19 4.190559  44 3.471818  243 2.722615 |
| 35 | NodeID Score  46 1.320278  68 1.320278  99 1.320278  131 1.320278  175 1.320278  177 1.320278  225 1.320278  227 1.320278  278 1.320278  321 1.320278 |

Results Table 3

# Evaluation of the recommendation system

## First Step:

We created the following function to evaluate the similarity percentage (number of common recommendation in the top 10 friends list) for every pair of functions (FoF-jaccard, Jaccard-Adamic, FoF-Adamic). In detail, the logic behind follows:

1. Find the 40 IDs that are multiple of 100 (excluded 0).
2. For each node of 40 and for every pair of functions, find the number of nodes that are common and divided by 10.
3. Calculate the average similarity between the pair of functions.

**R code used:**

#Id that is multiple of 100

filteredId**<-**unique**(**c**[**c**$**V1%%100**==**0,1**])**

#Exclude 0

filteredId**<-**filteredId**[!**filteredId%in%0**]**

fofjacsum**<-**0

jacadsum**<-**0

fofadsum**<-**0

#for each node out of the 40

**for** **(**i **in** filteredId**){**

ai**<-**friends**(**i, c**)**

fof**<-**fof\_fun**(**ai**[[**1**]]**, ai**[[**2**]])**

jac**<-**jaccard\_fun**(**ai**[[**1**]]**, ai**[[**2**]])**

adam**<-**ada**(**c, ai**[[**1**]]**, ai**[[**2**]])**

#similarity percentage for every pair

q1**<-**100**\***length**(**intersect**(**fof**$**NodeID, jac**$**NodeID**))/**length**(**fof**$**NodeID**)**

q2**<-**100**\***length**(**intersect**(**jac**$**NodeID, adam**$**NodeID**))/**length**(**jac**$**NodeID**)**

q3**<-**100**\***length**(**intersect**(**fof**$**NodeID, adam**$**NodeID**))/**length**(**fof**$**NodeID**)**

fofjacsum**<-**fofjacsum**+**q1

jacadsum**<-**jacadsum**+**q2

fofadsum**<-**fofadsum**+**q3

print**(**paste**(**"Node:",i,"Fof-Jaccard:", q1**))**

print**(**paste**(**"Node:",i,"Jaccard-Adamic:",q2**))**

print**(**paste**(**"Node:",i,"Fof-Adamic:", q3**))** **}**

#Average similarity between algorithms

print**(**paste**(**"Average Fof-Adamic:", fofjacsum**/**40**))**

print**(**paste**(**"Average Jaccard-Adamic:", jacadsum**/**40**))**

print**(**paste**(**"Average Fof-Adamic:", fofadsum**/**40**))**

Code Table 6

**Results:**

Following the similarity percentages for each node out of the 40.

|  |  |
| --- | --- |
| Similarity Percentages per node | |
| [1] "Node: 100 Fof-Jaccard: 20"  [1] "Node: 100 Jaccard-Adamic: 30"  [1] "Node: 100 Fof-Adamic: 50"  [1] "Node: 200 Fof-Jaccard: 90"  [1] "Node: 200 Jaccard-Adamic: 80"  [1] "Node: 200 Fof-Adamic: 90"  [1] "Node: 300 Fof-Jaccard: 60"  [1] "Node: 300 Jaccard-Adamic: 60"  [1] "Node: 300 Fof-Adamic: 100"  [1] "Node: 400 Fof-Jaccard: 80"  [1] "Node: 400 Jaccard-Adamic: 80"  [1] "Node: 400 Fof-Adamic: 100"  [1] "Node: 500 Fof-Jaccard: 90"  [1] "Node: 500 Jaccard-Adamic: 80"  [1] "Node: 500 Fof-Adamic: 90"  [1] "Node: 600 Fof-Jaccard: 10"  [1] "Node: 600 Jaccard-Adamic: 10"  [1] "Node: 600 Fof-Adamic: 100"  [1] "Node: 700 Fof-Jaccard: 30"  [1] "Node: 700 Jaccard-Adamic: 30"  [1] "Node: 700 Fof-Adamic: 90"  [1] "Node: 800 Fof-Jaccard: 90"  [1] "Node: 800 Jaccard-Adamic: 90"  [1] "Node: 800 Fof-Adamic: 100"  [1] "Node: 900 Fof-Jaccard: 70"  [1] "Node: 900 Jaccard-Adamic: 70"  [1] "Node: 900 Fof-Adamic: 90"  [1] "Node: 1000 Fof-Jaccard: 60"  [1] "Node: 1000 Jaccard-Adamic: 70"  [1] "Node: 1000 Fof-Adamic: 80"  [1] "Node: 1100 Fof-Jaccard: 80"  [1] "Node: 1100 Jaccard-Adamic: 80"  [1] "Node: 1100 Fof-Adamic: 100"  [1] "Node: 1200 Fof-Jaccard: 0"  [1] "Node: 1200 Jaccard-Adamic: 30"  [1] "Node: 1200 Fof-Adamic: 30"  [1] "Node: 1300 Fof-Jaccard: 50"  [1] "Node: 1300 Jaccard-Adamic: 40"  [1] "Node: 1300 Fof-Adamic: 90"  [1] "Node: 1400 Fof-Jaccard: 20"  [1] "Node: 1400 Jaccard-Adamic: 20"  [1] "Node: 1400 Fof-Adamic: 100"  [1] "Node: 1500 Fof-Jaccard: 70"  [1] "Node: 1500 Jaccard-Adamic: 70"  [1] "Node: 1500 Fof-Adamic: 100"  [1] "Node: 1600 Fof-Jaccard: 50"  [1] "Node: 1600 Jaccard-Adamic: 50"  [1] "Node: 1600 Fof-Adamic: 100"  [1] "Node: 1700 Fof-Jaccard: 10"  [1] "Node: 1700 Jaccard-Adamic: 10"  [1] "Node: 1700 Fof-Adamic: 90"  [1] "Node: 1800 Fof-Jaccard: 100"  [1] "Node: 1800 Jaccard-Adamic: 100"  [1] "Node: 1800 Fof-Adamic: 100"  [1] "Node: 2000 Fof-Jaccard: 70"  [1] "Node: 2000 Jaccard-Adamic: 70"  [1] "Node: 2000 Fof-Adamic: 100"  [1] "Node: 2100 Fof-Jaccard: 70"  [1] "Node: 2100 Jaccard-Adamic: 60"  [1] "Node: 2100 Fof-Adamic: 80" | [1] "Node: 2200 Fof-Jaccard: 90"  [1] "Node: 2200 Jaccard-Adamic: 90"  [1] "Node: 2200 Fof-Adamic: 100"  [1] "Node: 2300 Fof-Jaccard: 40"  [1] "Node: 2300 Jaccard-Adamic: 40"  [1] "Node: 2300 Fof-Adamic: 100"  [1] "Node: 2400 Fof-Jaccard: 10"  [1] "Node: 2400 Jaccard-Adamic: 20"  [1] "Node: 2400 Fof-Adamic: 60"  [1] "Node: 2500 Fof-Jaccard: 90"  [1] "Node: 2500 Jaccard-Adamic: 90"  [1] "Node: 2500 Fof-Adamic: 100"  [1] "Node: 2600 Fof-Jaccard: 100"  [1] "Node: 2600 Jaccard-Adamic: 100"  [1] "Node: 2600 Fof-Adamic: 100"  [1] "Node: 2700 Fof-Jaccard: 30"  [1] "Node: 2700 Jaccard-Adamic: 30"  [1] "Node: 2700 Fof-Adamic: 80"  [1] "Node: 2800 Fof-Jaccard: 90"  [1] "Node: 2800 Jaccard-Adamic: 90"  [1] "Node: 2800 Fof-Adamic: 100"  [1] "Node: 2900 Fof-Jaccard: 20"  [1] "Node: 2900 Jaccard-Adamic: 30"  [1] "Node: 2900 Fof-Adamic: 90"  [1] "Node: 3000 Fof-Jaccard: 70"  [1] "Node: 3000 Jaccard-Adamic: 70"  [1] "Node: 3000 Fof-Adamic: 90"  [1] "Node: 3100 Fof-Jaccard: 60"  [1] "Node: 3100 Jaccard-Adamic: 60"  [1] "Node: 3100 Fof-Adamic: 80"  [1] "Node: 3200 Fof-Jaccard: 50"  [1] "Node: 3200 Jaccard-Adamic: 50"  [1] "Node: 3200 Fof-Adamic: 100"  [1] "Node: 3300 Fof-Jaccard: 70"  [1] "Node: 3300 Jaccard-Adamic: 80"  [1] "Node: 3300 Fof-Adamic: 90"  [1] "Node: 3500 Fof-Jaccard: 60"  [1] "Node: 3500 Jaccard-Adamic: 70"  [1] "Node: 3500 Fof-Adamic: 90"  [1] "Node: 3600 Fof-Jaccard: 0"  [1] "Node: 3600 Jaccard-Adamic: 0"  [1] "Node: 3600 Fof-Adamic: 90"  [1] "Node: 3700 Fof-Jaccard: 20"  [1] "Node: 3700 Jaccard-Adamic: 20"  [1] "Node: 3700 Fof-Adamic: 100"  [1] "Node: 3800 Fof-Jaccard: 50"  [1] "Node: 3800 Jaccard-Adamic: 50"  [1] "Node: 3800 Fof-Adamic: 100"  [1] "Node: 3900 Fof-Jaccard: 90"  [1] "Node: 3900 Jaccard-Adamic: 90"  [1] "Node: 3900 Fof-Adamic: 100"  [1] "Node: 4000 Fof-Jaccard: 90"  [1] "Node: 4000 Jaccard-Adamic: 90"  [1] "Node: 4000 Fof-Adamic: 100"  [1] "Node: 1900 Fof-Jaccard: 10"  [1] "Node: 1900 Jaccard-Adamic: 20"  [1] "Node: 1900 Fof-Adamic: 80"  [1] "Node: 3400 Fof-Jaccard: 60"  [1] "Node: 3400 Jaccard-Adamic: 60"  [1] "Node: 3400 Fof-Adamic: 100" |

Results Table 4

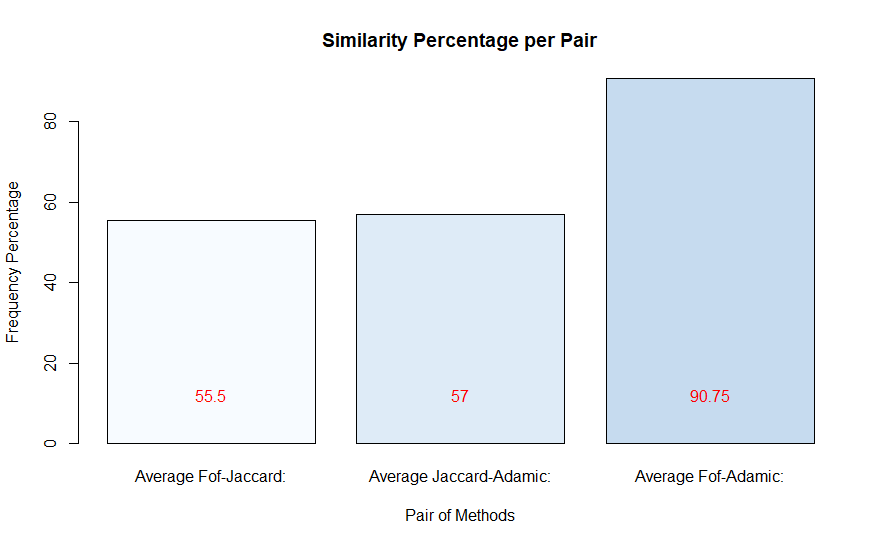


Figure 1

As we can notice from the aforementioned table, FoF and Adamic methods have higher similarity percentage (90,75%) whereas Jaccard is less similar to FoF and Adamic, 55,5 and 57% respectively.

## Second Step:

We created the following function to evaluate and forecast if the recommendations are going to be accepted by users. We evaluated all three recommendation systems using the same random choice and performed the above experiment 100 times. In detail, the logic behind follows:

1. Randomly choose a real friend connection; call the two friends F1 and F2.
2. Remove their friendship from the graph.
3. Compute friend recommendations for F1 and F2 (10 recommendations) for each function (FoF, Jaccard, Adamic) and keep the position that F2 appears in the list of F1 and the opposite (6 positions for all 3 functions).
4. For each similarity function average the respective two numbers.
5. Iterate the aforementioned steps 100 times.



count**<-**0

#vector for friends

llmrp**<-**c**()**

listina**<-**list**()**

**while** **(**count**<=**100**){**

print**(**count**)**

#Randomly choose a real friend connection

x1 **<-** sample**(**0**:**nrow**(**c**)**,1**)**

x2 **<-** which**(**c**[**x1,**]$**V1**==**c**$**V2 **&** c**[**x1,**]$**V2**==**c**$**V1**)[**1**]**

#Randomly choose a real friend connection

**while** **((**x1 %in% llmrp**)** **&** **(**x2 %in% llmrp**)){**

x1 **<-** sample**(**0**:**nrow**(**c**)**,1**)**

x2 **<-** which**(**c**[**x1,**]$**V1**==**c**$**V2 **&** c**[**x1,**]$**V2**==**c**$**V1**)[**1**]**

**}**

#add to vector

llmrp**<-**c**(**llmrp,x1,x2**)**

#Two friends F1, F2

F1**<-**c**[**x1,**]$**V1

F2**<-**c**[**x1,**]$**V2

#Remove the friendship from the graph(both directions)

y**<-**c**[-**c**(**x1, x2**)**,**]**

kik**<-**friends**(**F1, y**)**

#call fof

fo**<-**fof\_fun**(**kik**[[**1**]]**, kik**[[**2**]])**

#keep the position that F2 appears

positionfoF2**<-**which**(**fo**$**NodeID**==**F2**)[**1**]**

#call jaccard

jak**<-**jaccard\_fun**(**kik**[[**1**]]**, kik**[[**2**]])**

#keep the position that F2 appears

positionjakF2**<-**which**(**jak**$**NodeID**==**F2**)[**1**]**

#call adam

ad**<-**ada**(**y, kik**[[**1**]]**, kik**[[**2**]])**

#keep the position that F2 appears

positionadF2**<-**which**(**ad**$**NodeID**==**F2**)[**1**]**

kik2**<-**friends**(**F2, y**)**

#call fof

fo2**<-**fof\_fun**(**kik2**[[**1**]]**, kik2**[[**2**]])**

#keep the position that F1 appears

positionfo2F1**<-**which**(**fo2**$**NodeID**==**F1**)[**1**]**

#call jaccard

jak2**<-**jaccard\_fun**(**kik2**[[**1**]]**, kik2**[[**2**]])**

#keep the position that F1 appears

positionjak2F1**<-**which**(**jak2**$**NodeID**==**F1**)[**1**]**

#call adam

ad2**<-**ada**(**y, kik2**[[**1**]]**, kik2**[[**2**]])**

#keep the position that F1 appears

positionad2F1**<-**which**(**ad2**$**NodeID**==**F1**)[**1**]**

#if there is a value in every position evaluate all three reccommendation systems

**if** **(!**is.na**(**positionfoF2**)** **&** **!**is.na**(**positionjakF2**)** **&** **!**is.na**(**positionadF2**)** **&** **!**is.na**(**positionfo2F1**)** **&** **!**is.na**(**positionjak2F1**)** **&** **!**is.na**(**positionad2F1**)){**

#average two numbers for each similarity function

avgfof**<-(**positionfoF2**+**positionfo2F1**)/**2

avgjak**<-(**positionjakF2**+**positionjak2F1**)/**2

avgad**<-(**positionadF2**+**positionad2F1**)/**2

#save results to listina

listina**<-**list.append**(**listina, list**(**c**(**F1, "-" ,F2**)**, avgfof,avgjak, avgad**))**

#increase the counter

count**<-**count**+**1

**}**

**}**

summationfof**<-**0

summationjac**<-**0

summationada**<-**0

#calculate summation of scores per recommendation system

**for(** i **in** 1**:(**length**(**listina**))){**

summationfof**<-**summationfof**+**sum**(**listina**[[**i**]][[**2**]])**

summationjac**<-**summationjac**+**sum**(**listina**[[**i**]][[**3**]])**

summationada**<-**summationada**+**sum**(**listina**[[**i**]][[**4**]])**

**}**

#calculate the average rank per recommendation system

averagefof**<-**summationfof**/**100

averagejac**<-**summationjac**/**100

averageada**<-**summationada**/**100

Code Table 7

**Results:**

We run twice the experiment and we get the following results. In conclusion, there is consistency among the results. We can infer that the Jaccard method is better followed by Adamic and lastly by FoF.

* 1st run (100 iterations)

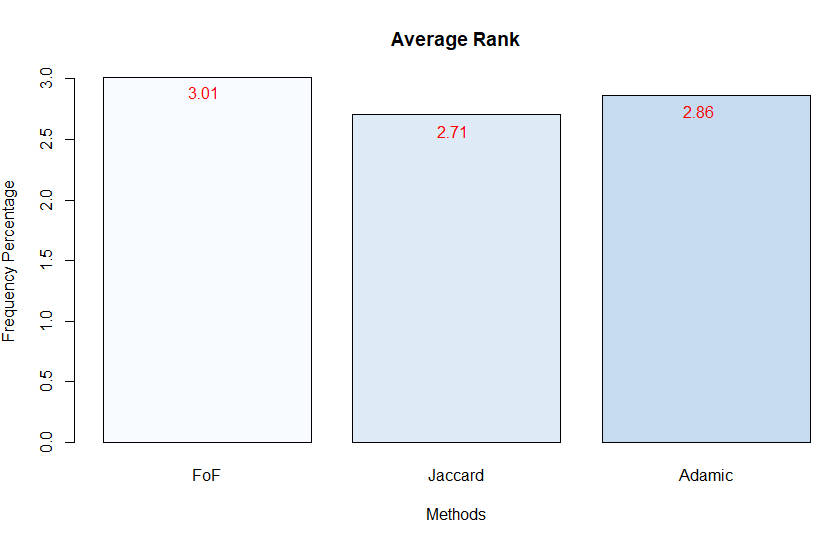


Figure 2

* 2nd run (100 iterations)

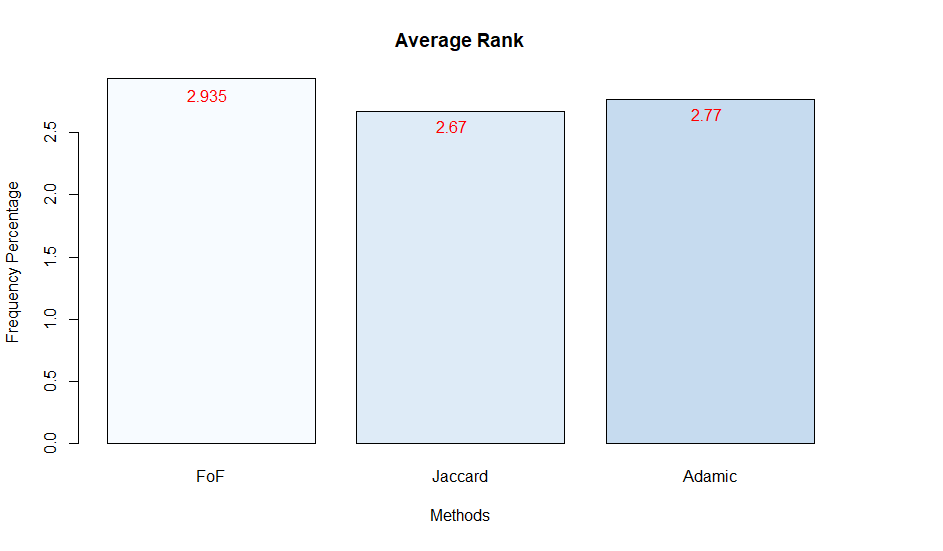


Figure 3

# Assignment Bonus

## Alternative Evaluation Method:

We created the following function to evaluate and forecast if the recommendations are going to be accepted by users. We evaluated all three recommendation systems using the same random choice and performed the above experiment 100 times. In detail, the logic behind follows:

1. Randomly choose a real friend connection; call the two friends F1 and F2.
2. Remove their friendship from the graph.
3. Compute friend recommendations for F1 and F2 (10 recommendations) for each function (FoF, Jaccard, Adamic) and keep the position that F2 appears in the list of F1 and the opposite (6 positions for all 3 functions).
4. For each similarity function calculate the **median** of the respective two numbers.
5. Iterate the aforementioned steps 100 times.

count**<-**0

#vector for friends

llmrp**<-**c**()**

listina**<-**list**()**

**while** **(**count**<**100**){**

print**(**count**)**

#Randomly choose a real friend connection

x1 **<-** sample**(**0**:**nrow**(**c**)**,1**)**

x2 **<-** which**(**c**[**x1,**]$**V1**==**c**$**V2 **&** c**[**x1,**]$**V2**==**c**$**V1**)[**1**]**

**while** **((**x1 %in% llmrp**)** **&** **(**x2 %in% llmrp**)){**

x1 **<-** sample**(**0**:**nrow**(**c**)**,1**)**

x2 **<-** which**(**c**[**x1,**]$**V1**==**c**$**V2 **&** c**[**x1,**]$**V2**==**c**$**V1**)[**1**]}**

#add to vector

llmrp**<-**c**(**llmrp,x1,x2**)**

#Two friends F1, F2

F1**<-**c**[**x1,**]$**V1

F2**<-**c**[**x1,**]$**V2

#Remove the friendship from the graph(both directions)

y**<-**c**[-**c**(**x1, x2**)**,**]**

#y<-facebook\_sample[-c(x1, which(facebook\_sample[x1,]$V1==facebook\_sample$V2 & facebook\_sample[x1,]$V2==facebook\_sample$V1)[1]),]

kik**<-**friends**(**F1, y**)**

kik**[[**1**]]**

fo**<-**fof\_fun**(**kik**[[**1**]]**, kik**[[**2**]])**

positionfoF2**<-**which**(**fo**$**NodeID**==**F2**)[**1**]**

positionfoF2

jak**<-**jaccard\_fun**(**kik**[[**1**]]**, kik**[[**2**]])**

positionjakF2**<-**which**(**jak**$**NodeID**==**F2**)[**1**]**

positionjakF2

ad**<-**ada**(**y, kik**[[**1**]]**, kik**[[**2**]])**

positionadF2**<-**which**(**ad**$**NodeID**==**F2**)[**1**]**

kik2**<-**friends**(**F2, y**)**

kik2**[[**1**]]**

fo2**<-**fof\_fun**(**kik2**[[**1**]]**, kik2**[[**2**]])**

positionfo2F1**<-**which**(**fo2**$**NodeID**==**F1**)[**1**]**

positionfo2F1

jak2**<-**jaccard\_fun**(**kik2**[[**1**]]**, kik2**[[**2**]])**

positionjak2F1**<-**which**(**jak2**$**NodeID**==**F1**)[**1**]**

positionjak2F1

ad2**<-**ada**(**y, kik2**[[**1**]]**, kik2**[[**2**]])**

positionad2F1**<-**which**(**ad2**$**NodeID**==**F1**)[**1**]**

positionad2F1

**if** **(!**is.na**(**positionfoF2**)** **&** **!**is.na**(**positionjakF2**)** **&** **!**is.na**(**positionadF2**)** **&** **!**is.na**(**positionfo2F1**)** **&** **!**is.na**(**positionjak2F1**)** **&** **!**is.na**(**positionad2F1**)){**

#The change for the alternative evaluation method is here!!!!!!!!!!!!!!!!!

medianfof**<-**median**(**c**(**positionfoF2,positionfo2F1**))**

medianjak**<-**median**(**c**(**positionjakF2,positionjak2F1**))**

medianad**<-**median**(**c**(**positionadF2,positionad2F1**))**

listina**<-**list.append**(**listina, list**(**c**(**F1, "-" ,F2**)**, medianfof,medianjak, medianad**))**

count**<-**count**+**1**}** **}**

summationfof**<-**0

summationjac**<-**0

summationada**<-**0

**for(** i **in** 1**:(**length**(**listina**))){**

summationfof**<-**summationfof**+**sum**(**listina**[[**i**]][[**2**]])**

summationjac**<-**summationjac**+**sum**(**listina**[[**i**]][[**3**]])**

summationada**<-**summationada**+**sum**(**listina**[[**i**]][[**4**]])**

**}**

averagefof**<-**summationfof**/**100

averagejac**<-**summationjac**/**100

averageada**<-**summationada**/**100

Code Table 8

**Results:**

We run the experiment and get the following graph. We can infer that the Jaccard method is better followed by Adamic and lastly by FoF.

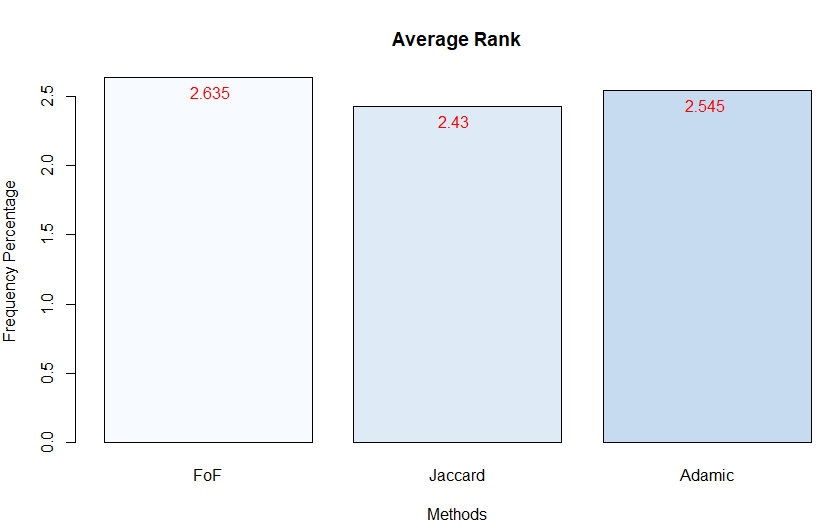


Figure 4

## Additional Scoring Function:

**Hub Promoted Index** is a measure defined as the ratio of common neighbors of nodes a and b to the minimum of degrees of nodes a and b. Hub Promoted index SHPI(a,b) is computed as: SHPI(a,b) = |Γ(a)∩Γ(b)| min{ka,kb}.

**R code used:**

hub**<-function(**c, target, lista**){**

#saves the common friends of those nodes that are not friends of the target

**for** **(**i **in** 1**:**length**(**lista**)){**

lista**[[**i**]]<-**list.append**(**lista**[[**i**]]**, list**(**intersect**(**target, lista**[[**i**]]$**V2**)))**

**}**

listq4**<-**list**()**

#For every non friend of target

**for** **(**i **in** 1**:**length**(**lista**)){**

vectorr**<-**c**()**

omg**<-**length**(**lista**[[**i**]][[**3**]][[**1**]])**

#For every common friend that the target has with the non friend, calculate the number of friends

#that it has, take its 1/logarithm and save it.

**for** **(**j **in** 1**:**length**(**unlist**(**lista**[[**i**]][[**3**]]))){**

vectorr**<-**c**(**vectorr, length**(**c**[**c**$**V1**==**lista**[[**i**]][[**3**]][[**1**]][**j**]**,2**]))**

**}**

minimum**<-**omg**/**min**(**vectorr**)**

listq4**<-**list.append**(**listq4, list**(**unique**(**lista**[[**i**]][[**1**]])**, minimum**))**

**}**

listb**<-**head**(**listq4**[**order**(**sapply**(**listq4, **function(**x**)** x**[[**2**]]**, simplify**=TRUE)**, decreasing**=TRUE)]**,11**)**

#formats the results

df**<-**data.frame**()**

**for** **(**i **in** 1**:**length**(**listb**)){**

df**<-**rbind**(**df, t**(**data.frame**(**unlist**(**listb**[[**i**]]))))**

**}**

rownames**(**df**)<-NULL**

names**(**df**)<-**c**(**"NodeID","Score"**)**

**if** **(**df**[**nrow**(**df**)**,2**]==**df**[**nrow**(**df**)-**1,2**]){**

**if** **(**df**[**nrow**(**df**)**,1**]==**max**(**df**[**nrow**(**df**)**,1**]**, df**[**nrow**(**df**)-**1,1**])){**

df**<-**df**[-**nrow**(**df**)**,**]**

**}**

**else{**

df**<-**df**[-(**nrow**(**df**)-**1**)**,**]**

**}**

**}**

**else** **{**

df**<-**df**[-**nrow**(**df**)**,**]**

**}**

return**(**df**)**

**}**

**Calling with parameters friends list of target and the friends of the non-friends of target:**

hub**(**c, a107**[[**1**]]**, a107**[[**2**]])**

hub**(**c, a1126**[[**1**]]**, a1126**[[**2**]])**

hub**(**c, a14**[[**1**]]**, a14**[[**2**]])**

hub**(**c, a35**[[**1**]]**, a35**[[**2**]])**

**Changes in evaluation code:**

count**<-**0

#vector for friends

llmrp**<-**c**()**

listina**<-**list**()**

**while** **(**count**<=**100**){**

print**(**count**)**

#Randomly choose a real friend connection

x1 **<-** sample**(**0**:**nrow**(**c**)**,1**)**

x2 **<-** which**(**c**[**x1,**]$**V1**==**c**$**V2 **&** c**[**x1,**]$**V2**==**c**$**V1**)[**1**]**

**while** **((**x1 %in% llmrp**)** **&** **(**x2 %in% llmrp**)){**

x1 **<-** sample**(**0**:**nrow**(**c**)**,1**)**

x2 **<-** which**(**c**[**x1,**]$**V1**==**c**$**V2 **&** c**[**x1,**]$**V2**==**c**$**V1**)[**1**]**

**}**

#add to vector

llmrp**<-**c**(**llmrp,x1,x2**)**

#Two friends F1, F2

F1**<-**c**[**x1,**]$**V1

F2**<-**c**[**x1,**]$**V2

#Remove the friendship from the graph(both directions)

y**<-**c**[-**c**(**x1, x2**)**,**]**

kik**<-**friends**(**F1, y**)**

#call fof

fo**<-**fof\_fun**(**kik**[[**1**]]**, kik**[[**2**]])**

#keep the position that F2 appears

positionfoF2**<-**which**(**fo**$**NodeID**==**F2**)[**1**]**

#call jaccard

jak**<-**jaccard\_fun**(**kik**[[**1**]]**, kik**[[**2**]])**

#keep the position that F2 appears

positionjakF2**<-**which**(**jak**$**NodeID**==**F2**)[**1**]**

#call adam

ad**<-**ada**(**y, kik**[[**1**]]**, kik**[[**2**]])**

#keep the position that F2 appears

positionadF2**<-**which**(**ad**$**NodeID**==**F2**)[**1**]**

#call hub

hubu**<-**hub**(**y, kik**[[**1**]]**, kik**[[**2**]])**

#keep the position that F2 appears

positionhubF2**<-**which**(**hubu**$**NodeID**==**F2**)[**1**]**

kik2**<-**friends**(**F2, y**)**

#call fof

fo2**<-**fof\_fun**(**kik2**[[**1**]]**, kik2**[[**2**]])**

#keep the position that F1 appears

positionfo2F1**<-**which**(**fo2**$**NodeID**==**F1**)[**1**]**

#call jaccard

jak2**<-**jaccard\_fun**(**kik2**[[**1**]]**, kik2**[[**2**]])**

#keep the position that F1 appears

positionjak2F1**<-**which**(**jak2**$**NodeID**==**F1**)[**1**]**

#call adam

ad2**<-**ada**(**y, kik2**[[**1**]]**, kik2**[[**2**]])**

#keep the position that F1 appears

positionad2F1**<-**which**(**ad2**$**NodeID**==**F1**)[**1**]**

#call hub

hub2**<-**hub**(**y, kik2**[[**1**]]**, kik2**[[**2**]])**

#keep the position that F1 appears

positionhub2F1**<-**which**(**hub2**$**NodeID**==**F1**)[**1**]**

#if there is a value in every position evaluate all three reccommendation systems

**if** **(!**is.na**(**positionhub2F1**)** **&** **!**is.na**(**positionhubF2**)** **&** **!**is.na**(**positionfoF2**)** **&** **!**is.na**(**positionjakF2**)** **&** **!**is.na**(**positionadF2**)** **&** **!**is.na**(**positionfo2F1**)** **&** **!**is.na**(**positionjak2F1**)** **&** **!**is.na**(**positionad2F1**)){**

#average two numbers for each similarity function

avgfof**<-(**positionfoF2**+**positionfo2F1**)/**2

avgjak**<-(**positionjakF2**+**positionjak2F1**)/**2

avgad**<-(**positionadF2**+**positionad2F1**)/**2

avghub**<-(**positionhubF2**+**positionhub2F1**)/**2

#save results to listina

listina**<-**list.append**(**listina, list**(**c**(**F1, "-" ,F2**)**, avgfof,avgjak, avgad, avghub**))**

#increase the counter

count**<-**count**+**1

**}**

**}**

summationfof**<-**0

summationjac**<-**0

summationada**<-**0

summationhub**<-**0

#calculate summation of scores per recommendation system

**for(** i **in** 1**:(**length**(**listina**))){**

summationfof**<-**summationfof**+**sum**(**listina**[[**i**]][[**2**]])**

summationjac**<-**summationjac**+**sum**(**listina**[[**i**]][[**3**]])**

summationada**<-**summationada**+**sum**(**listina**[[**i**]][[**4**]])**

summationhub**<-**summationhub**+**sum**(**listina**[[**i**]][[**5**]])**

**}**

#calculate the average rank per recommendation system

averagefof**<-**summationfof**/**100

averagejac**<-**summationjac**/**100

averageada**<-**summationada**/**100

averagehub**<-**summationhub**/**100

Code Table 9

**Results:**

The first 10 recommendations and their corresponding score after running the Hub method are listed below per the target node ID requested.

| nodeID | Recommendation || Score |
| --- | --- |
| 107 | NodeID Score  173 0.8000000  513 0.7600000  400 0.7200000  500 0.6800000  465 0.5600000  1912 0.5000000  630 0.4545455  514 0.4411765  524 0.4411765  394 0.4000000 |
| 1126 | NodeID Score  1172 3.296296  1750 2.750000  1811 2.722222  1845 2.423077  916 2.260000  1416 2.222222  1538 2.148148  1123 2.120000  1230 1.960000  1004 1.880000 |
| 14 | NodeID Score  17 0.6923077  140 0.6923077  2 0.6428571  111 0.6153846  137 0.5384615  162 0.5000000  333 0.4285714  19 0.3529412  44 0.3333333  243 0.3076923 |
| 35 | NodeID Score  46 0.1666667  68 0.1666667  99 0.1666667  131 0.1666667  175 0.1666667  177 0.1666667  225 0.1666667  227 0.1666667  278 0.1666667  321 0.1666667 |

Results Table 5

**Results:**

We run the experiment and get the following graph. We can infer that the Jaccard method is still better followed by Adamic, FoF and lastly by Hub.

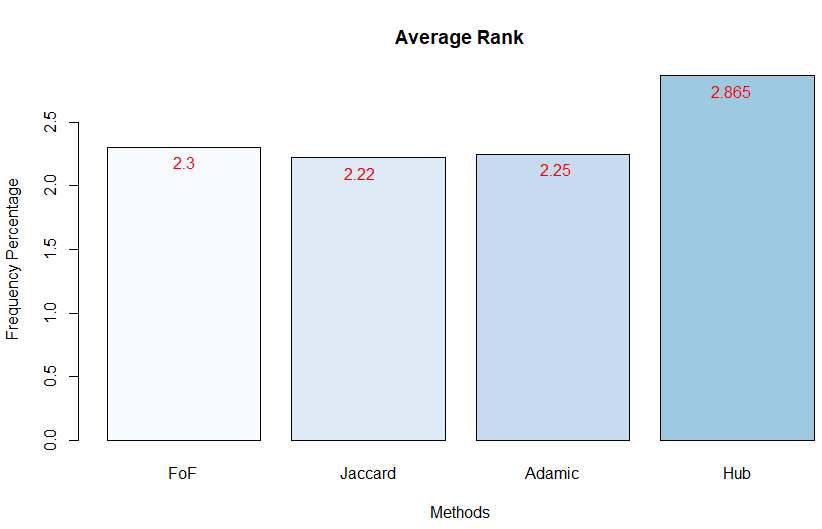


Figure 5