**FACEBOOK FRIEND RECOMMENDATION SYSTEM**

**Individual Project :**

***Student Id : 012443886***

***Student Name : Deepika Yannamani***

**Course ID : CMPE256**

**Course Name : Large Scale Analytics**

**Professor: Shih Yu Chang**

**Introduction**

A recommender system or a recommendation system is a subclass of information filtering system that seeks to predict the "rating" or "preference" that a user would give to an item. There are multiple approaches to solve this problem,Content based,Knowledge based, Colloborative filtering methods. Collaborative filtering says that, if your past behaviour/preferences were similar to some other users, then your future behaviour may be as well. In friend recommendation systems, the dataset contains people and their connections or friendships with other people. These connections are used as edges and people are used as nodes in a graph. The graph obtained from the dataset is called social network of friends. For a user, the recommendation system predicts the most suitable candidate for becoming friend of the user.

**Objective**

In this project, I am implementing a collaborative filtering recommendation system for suggesting friends on Facebook. I implemented two mechanisms for recommending a new friend in a social network. For user X, listed some non-friends in order, starting with the best friend recommendation and ending with the worst. A non-friend is a user who is not X and is not a friend of X. Depending on the recommendation algorithm, the list may include all non-friends or some of them. Further, the recommendations might not be symmetric.

**Innovation component in the project**

In this project, the uniqueness is the calculation of lonely nodes. Lonely nodes are the nodes in the social network who are not at all the best recommendations to anyone in the social network. The clustering coefficient is an important factor which determines the loneliness of a node in a social network. The clustering coefficient is the measure of how much the friends of a user are connected to each other. The implementation says that the lonely nodes are the nodes with least clustering coefficients. The results made from this observation is that a person is not lonely because he has no or less friends but due to the reason that his friends are not friends among themselves.

**Implementation**

*Software Requirements Jupyter Notebook*(Python 3.6)

*Hardware Requirements:* Any computer with Anaconda installed would work.

*Methodology:*

The methodology of this project consists of two core methods of recommendation. The first one is the *recommend\_by\_common\_friends* and *recommend\_by\_influence.*

*Recommend by common friends:*

If non-friend Y is your friend's friend, then maybe Y should be your friend too. If person Y is the friend of many of your friends, then Y is an even better recommendation. The best friend recommendation is the person with whom you have the largest number of mutual friends.

*Recommend by influence:*

*Consider the following hypothetical situation.*

Two of X’s friends are Y and Z. Y has only two friends (X and one other person). Z has 7 billion friends.Y and Z have no friends in common(besides X)

Since Y is highly selective in terms of friendship, and is a friend of X, X is likely to have a lot in common with Y’s other friend. On the other hand, Z is indiscriminate and there is little reason to believe that X should be friendly with any particular one of Z’s other friends. Incorporate the above idea into your friend recommendation algorithm. We can call the technique “influence scoring”. Suppose that user1 and user2 have three friends in common: f1, f2, and f3. In this problem, the score for user2 as a friend of user1 is 1/numfriends(f1) + 1/numfriends(f2) + 1/numfriends(f3), where numfriends(f) is the number of friends that f has. In other words, each friend F of user1 has a total influence score of 1 to contribute, and divides it equally among all of F's friends.

Algorithm steps

1. Two nodes are chosen at random.
2. Their friendship is removed from the graph.
3. Friend recommendations for F1 and F2 are computed.
4. Rank of F1 in F2's list of recommended friends is calculated.  
      
   Rank of F2 in F1's list of recommended friends is calculated.  
      
   Average of both rank is computed.
5. Friendship is put back to the graph.

For a perfect recommendation system, the first recommendation for F1 would be F2, and the first recommendation for F2 would be F1. In general, the closer to the front of the list these recommendations are, the better the recommendation system. For a good recommendation system, the average rank should be small.

**Datasets used**

Dataset is taken from SNAP (Stanford Network Analysis Project)  
 This dataset consists of 'circles' from Facebook. Facebook data was collected from survey participants using this Facebook app. The dataset includes node features (profiles), circles, and ego networks. Facebook data has been anonymized by replacing the Facebook-internal ids for each user with a new value.

**Dataset Analysis Report:**

*Type*: Graph

*Number of nodes*: 4039

*Number of edges*: 88234

*Average degree*: 43.6910

The Graph is connected

The graph is not directed

*Average clustering coefficient is*: 0.6055467186200876  
   
   
**Code**

Few important methods are added here. Full code is available in github :

# -\*- coding: utf-8 -\*-

"""

Created on Mon July 4 22:15:18 2019

@author: Deepika Yannamani

"""

import random

import networkx as nx

import matplotlib.pyplot as plt

from operator import itemgetter

G2 = nx.read\_edgelist('Facebook\_Dataset.txt', create\_using = nx.Graph(), nodetype = int)

def common\_friends(graph, user1, user2):

x1 = friends(graph, user1)

x2 = friends(graph, user2)

return set(x1&x2)

def number\_of\_common\_friends\_map(graph, user):

new\_dict = dict()

for each in graph.nodes():

if(each!=user):

if(each not in graph.neighbors(user)):

new\_dict[each] = len(common\_friends(graph,each,user))

return new\_dict

def number\_map\_to\_sorted\_list(map):

map = sorted(map.items(), key = itemgetter(1), reverse=True)

return map

def recommend\_by\_number\_of\_common\_friends(graph, user):

diction = dict()

diction = number\_of\_common\_friends\_map(graph,user)

diction = number\_map\_to\_sorted\_list(diction)

recommendations = []

for i in range(0,10):

recommendations.append(diction[i])

return recommendations

def calc\_score(graph, user, each):

score = 0

common = common\_friends(graph, user, each)

for item in common:

score = score + 1/(len(friends(graph, item)))

return score

def influence\_map(graph, user):

influence\_scores = dict()

for each in graph.nodes():

if(each != user):

score = calc\_score(graph, user, each)

influence\_scores[each] = score

return influence\_scores

def recommend\_by\_influence(graph, user):

recommendations = []

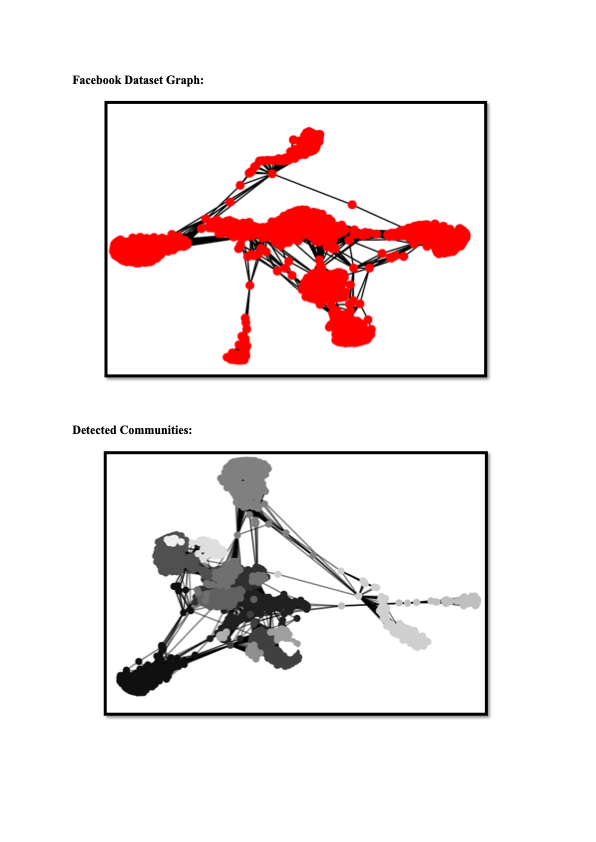
d=influence\_map(graph,user)

d = sorted(d.items(), key = itemgetter(1), reverse=True)

for i in range(0,10):

recommendations.append(d[i])

return recommendations

****

**Results and discussion**

***Recommendations for node 1222:***

***Recommendations by Method 1:***

[(1376, 97), (1833, 91), (1746, 88), (993, 86), (1390, 86), (1391, 83), (1714, 83), (1059, 81), (1516, 81), (1612, 81)]

***Recommendation by Method 2:***

[(107, 1.5041943556371893), (1888, 1.0776933333350602), (1352, 1.0612043383752092), (1377, 1.0363770157940952), (1730, 1.0127165834130822), (1663, 1.0070598176447842), (1551, 0.9766386140683305), (1813, 0.9717048924955747), (1768, 0.9555284637663646), (1199, 0.9479967116267918)]

***Average Rank of Method 1:***

4.0

***Average rank of Method 2***:

0

***Number of Same recommendations from both methods:*** 104

***Number of different recommendations from both methods:*** 3935

***Lonely Node:***

9

***Suicidal Tendency Nodes:***

(11, 0.0)

I conclude by saying that the method of recommendation by influence is better and more efficient than method of recommendation by number of common friends. This reference is drawn due to the values of average rank values of both methods. The average rank of influence method is smaller than average rank of common friend method. Also, the nodes with least clustering coefficient are called lonely nodes and have more tendency to be lonely and have suicidal tendency. I also say that the next possible friend is dependent on the influence score of current friends.

**References**

[1] https://courses.cs.washington.edu/courses/cse140/12su/homework/hw4/homework4.html [2] <https://en.wikipedia.org/wiki/Recommender_system>

[2]

http://www.gjaet.com/wp-content/uploads/2015/10/A-FRIEND-RECOMMENDATION- SYSTEM-FOR-SOCIAL-NETWORKS.pdf