FRIEND RECOMMENDATION SYSTEM IN PYTHON

Team Members:
15BCE0857 (Sushabh Deshmukh)
15BCE0137 (Srishti Sabharwal)

Report submitted for the
Final Project Review of

Course Code: CSE3021 – Social and Information Networks

Slot: A2 + TA2

Professor: Dr. Ilanthenral K P S K

1. 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. Recommender systems have become increasingly popular in recent years, and are utilized in a variety of areas including movies, music, news, books, research articles, search queries, social tags, and products in general. There are also recommender systems for experts, collaborators, jokes, restaurants, garments, financial services, life insurance, online dating, and Twitter pages. Facebook suggests people you could be friends with. The actual algorithms used by these companies are closely-guarded trade secrets. There are two general approaches: collaborative filtering and content-based filtering.

- **Collaborative filtering** says that, if your past behaviour/preferences were similar to some other users, then your future behaviour may be as well. As a concrete example, suppose that you like John, Paul, and George, and other people like John, Paul, George, and Ringo. Then it stands to reason that you will like Ringo as well, even if you had never previously heard of him. The recommender system does not have to understand anything about what “John”, “Paul”, “George”, and “Ringo” are — they could even be brands of toilet paper, and the algorithm would work identically.
- **Content-based filtering** considers the characteristics of the things you like, and it recommends similar sorts of things. For instance, if you like “Billie Jean”, “Crazy Train”, and “Don't Stop the Music”, then you might like other songs in the key of F-sharp minor, such as Rachmaninoff's “Piano Concerto No. 1”, even if no one else has ever had that particular set of favourite songs before.

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.

2. Literature Review Summary Table

Authors and Year	***Title***	***Concept***	***Methodology***	***Datasets***	***Relevant Finding***	***Limitations/Future Work***
Jeff Naruchitparames, Mehmet Hadi Gunes and Sushil Louis (2013)	*Friend Recommendation in Social Networks using Genetic Algorithms and Network Topology*	*Complex network theory, cognitive theory and Pareto-optimal genetic algorithm.*	*Social genome using Pareto optimal genetic algorithm.*	*Facebook*	*Social genome, Pareto optimal genetic algorithm*	*Not many features.*
Jyoti Sharma, Pinky Tanwar (2016)	*A Survey on Friend Recommendation System*	*Different Analysis*	*Detailed survey of latest literature*	*Ecommerce, publicly available*	*Pros and Cons of various recommenders*	*Lack of practicality*
Namrata Eklaspur, Anand Pashupathinath (2015)	*A Friend recommender system for social networks by life style extraction using probabilistic methods*	*Based on life style of users*	*Text Mining*	*Facebook*	*Feature extraction, text mining*	*Not standalone. Not many datasets involved.*

Chaitali Hegshetye, Namrata Bandbe, Yogesh Gajmal (2015)	*Survey on Friend Recommendation system for Social Networks*	*Semantic based recommender*	*Overview, 1.5 clique extension method*	*Publicly available*	*LJ crawler, ECS algorithm, 1.5 clique extension*	*No implementation.*
Zhibo Wang, Jilong Liao (2014)	*Friendbook: A Semantic based Friend recommendation system*	*Based on lifestyle and not social graphs*	*Similarity metric*	*Sensor rich smartphones*	*Friend matching graph*	*More sensors*

3. Objective of the project:

In this project, we implemented a collaborative filtering recommendation system for suggesting friends on Facebook. We implemented two mechanisms for recommending a new friend in a social network. For user X, we 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. We implemented a code in python that produces friend recommendations for U, in order from best to worst. This is done by assigning each potential friend a number called a score, where higher scores indicate a better match. Then the list is returned according to the score. Given user X, if two people Y and Z would be equally good as new friends for X (they have the same score), then they are listed in alphabetical order (for names) or numerical order (for numerical user IDs).

4. 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. The discussion follows that the nodes with low clustering coefficient and are not at all recommended by the recommendation system have more suicidal tendency.

The results also show that every other friend of a user has an influence in the life of user and the next friend of the user is largely dependent on current friends of the user. The recommendation results for influence method show that the every new friendship is influenced by the current friendships.

5. Work done and implementation

a. Methodology adapted:

Software Requirements:

Spyder (Python 3.6)

Networkx (Python Library)

Hardware Requirements:

Any computer with python 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 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.

Evaluation:

The following algorithm is performed 100 times on the dataset and average is calculated of all average values returned to evaluate the results

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.

b. Dataset used :

a. Dataset is taken from SNAP (Stanford Network Analysis Project)
   Source (Citation): J. McAuley and J. Leskovec. Learning to Discover Social Circles in Ego Networks. NIPS, 2012.

   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.

b. The project is based on CSE140 course at University of Washington named "Social Networking and Recommendation Systems" *https://courses.cs.washington.edu/courses/cse140/12su/homework/hw4/homework4.html*

c. Tools Used:

Spyder (Python 3.4) is used to write the code. Networkx library of python is used to create and analyse the dataset. NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. It has the following features:

- Data structures for graphs, digraphs, and multigraphs
- Many standard graph algorithms
- Network structure and analysis measures
- Generators for classic graphs, random graphs, and synthetic networks
- Nodes can be "anything" (e.g., text, images, XML records)
- Edges can hold arbitrary data (e.g., weights, time-series)
- Well tested with over 90% code coverage
- Additional benefits from Python include fast prototyping, easy to teach, and multi-platform

d. Screenshots and Demo

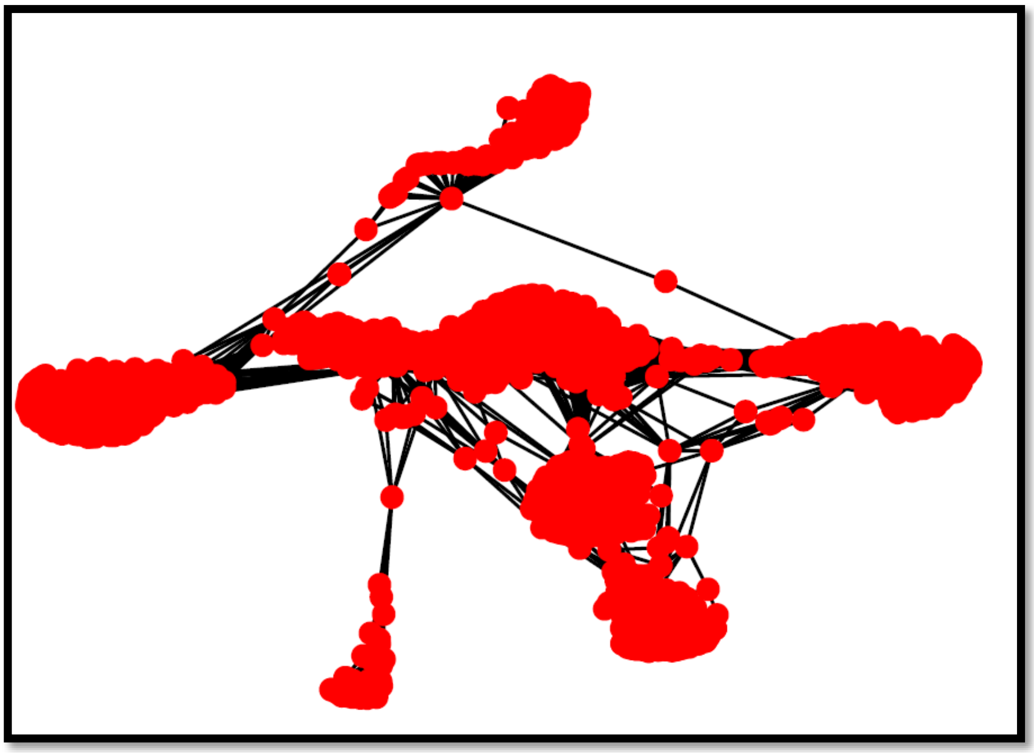
```
# -*- coding: utf-8 -*-
"""
Created on Mon Sep  4 22:15:18 2017
@author: Sushabh n' Srishti
"""
```

```
# Names: Sushabh Deshmukh
#       Srishti Sabharwal
# CSE 3021 Social Networks
# J Component


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 draw():
    nx.draw(G)
    plt.savefig("facebook.pdf")
    plt.show()


def friends(graph, user):
    return set(graph.neighbors(user))


def friends_of_friends(graph, user):
    x=[]
    for each in graph.neighbors(user):
        for item in graph.neighbors(each):
            x.append(item)
    return set(x)

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
```

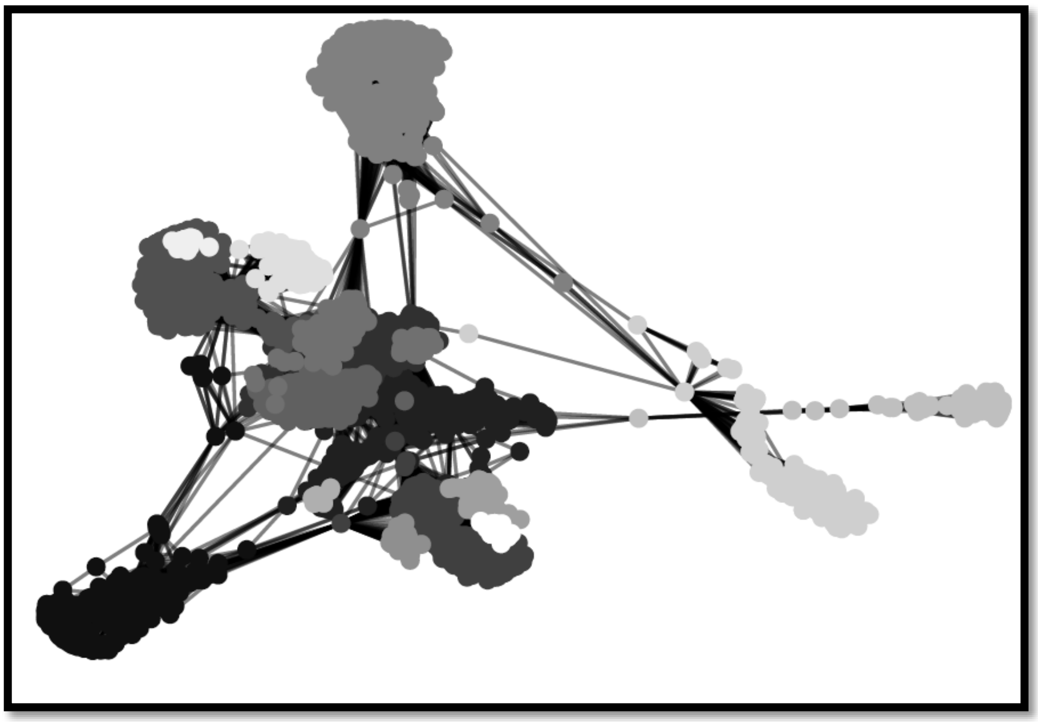
```
def number_map_to_sorted_list(map):
    map = sorted(map.items(), key = itemgetter(1), reverse=True)
    return map


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


def return_pure_list(recommendations):
    pure_list = []
    for each in recommendations:
        pure_list.append(each[0])
    return pure_list
```

```
def compute_avg_rank(G):
    avg=0
    AVG=0
    l=[]
    for i in range(0,1000):
        f1 = random.choice(G.nodes())
        f2 = random.choice(G.nodes())
        if(f1!=f2):
            if(G.has_edge(f1,f2)):
                G.remove_edge(f1,f2)
                l1 = recommend_by_number_of_common_friends(G,f1)
                l2 = recommend_by_number_of_common_friends(G,f2)
                L1 = recommend_by_influence(G,f1)
                L2 = recommend_by_influence(G,f2)
                if   f1   in   return_pure_list(l2)   and   f2   in
return_pure_list(l1):
                    r1 = return_pure_list(l2).index(f1)
                    r2 = return_pure_list(l1).index(f2)
                    avg=avg+(r1+r2)/2
                if   f1   in   return_pure_list(L2)   and   f2   in
return_pure_list(L1):
                    R1 = return_pure_list(L2).index(f1)
                    R2 = return_pure_list(L1).index(f2)
                    AVG=AVG+(R1+R2)/2
        G.add_edge(f1,f2)
    l.append(avg)
    l.append(AVG)
    print("Average Rank of Method 1: ")
    print(l[0])
    print("Average rank of Method 2: ")
    print(l[1])
    return l



def same_and_different_recommendations(graph):
    res=[]
    count_same, count_diff = 0,0
    for i in range(0,4039):
        l1,l2 = [], []
        l1                                                         =
return_pure_list(recommend_by_number_of_common_friends(graph, i))
        l2 = return_pure_list(recommend_by_influence(graph, i))
        if(l1 == l2):
            count_same = count_same + 1
        else:
            count_diff = count_diff +1
    res.append(count_same)
    res.append(count_diff)
    print("Number of Same recommendations from both methods: ")
    print(res[0])
    print("Number of different recommendations from both methods: ")
    print(res[1])
    return res

def most_common(lst):
    return max(set(lst), key=lst.count)

def predict_lonely_nodes(graph):
    res_2=[]
    for i in range(0,graph.number_of_nodes()):
        l_2 = return_pure_list(recommend_by_number_of_common_friends(graph,i))
        res_2.append(l_2[9])
    lonely_2 = most_common(res_2)
    print("Lonely Node: ")
    print(lonely_2)
    return lonely_2



def suicidal_tendency_nodes(graph):
    ccs = nx.clustering(graph)
    print("Suicidal Tendency Nodes: ")
    print(min(ccs.items(), key=lambda x: x[1]))
    return min(ccs.items(), key=lambda x: x[1])


def show(graph, user):
    nx.draw_networkx_nodes(graph,pos=nx.spring_layout(graph),

nodelist=return_pure_list(recommend_by_influence(graph,user)),
                       node_color='b',
                       node_size=500,
                   alpha=0.8)
    plt.savefig('x.pdf')
    print("Recommendation by Method 1: " )
    print(recommend_by_influence(graph, user))
    print("Recommendation by Method 2: ")
    print(recommend_by_number_of_common_friends(graph, user))

show(G2,1222 )
compute_avg_rank(G2)
same_and_different_recommendations(G2)
predict_lonely_nodes(G2)
suicidal_tendency_nodes(G2)
```

Facebook Dataset Graph:



Detected Communities:



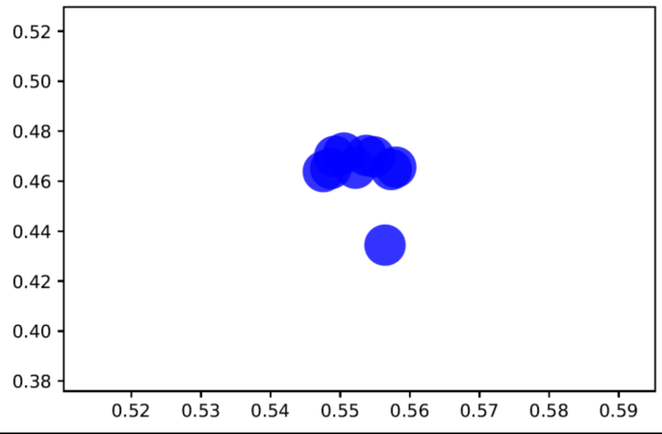
Dataset Analysis Report:

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

6. 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)



We 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. We also say that the next possible friend is dependent on the influence score of current friends.

7. References

[1] https://courses.cs.washington.edu/courses/cse140/12su/homework/hw4/homework4.html

[2] https://en.wikipedia.org/wiki/Recommender_system

[3] https://networkx.github.io/documentations

[4] https://snap.stanford.edu/data/egonets-Facebook.html

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

[6] Jeff Naruchitparames, Mehmet Hadi Gunes and Sushil Louis (2013) "Friend Recommendation in Social Networks using Genetic Algorithms and Network Topology"