Social Networks Analysis : Link Prediction

By

ALFAIZ ULLAH (11700114002)

SUBHADHRITI MAIKAP (11700114077)

ARITRA DAS (11700114011)

PRINCE SINHA (11700114007)

UNDER THE GUIDANCE OF

Prof. SK. Mazharul Islam

PROJECT REPORT SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING

RCC INSTITUTE OF INFORMATION TECHNOLOGY

Session 2017-2018

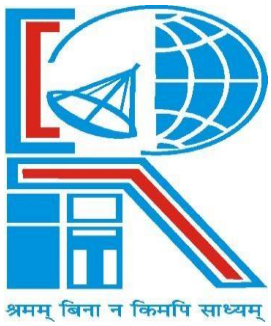
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

RCC INSTITUTE OF INFORMATION TECHNOLOGY

[Affiliated to West Bengal University of Technology]
CANAL SOUTH ROAD, BELIAGHATA, KOLKATA-700015

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

RCC INSTITUTE OF INFORMATION TECHNOLOGY



TO WHOM IT MAY CONCERN

I hereby recommend that the Project entitled Social Networks Analysis : Link Prediction & Community Detection prepared under my supervision by **ALFAIZ ULLAH (Reg. No. –141170110002,Class Roll No. –CSE/2014/039),ARITRA DAS(Reg. No. –141170110011,Class Roll No.CSE/2014/057),SUBHODHRITI MAIKAP(Reg. No. –141170110077,Class Roll No. –CSE/2014/059),PRICE SINHA(Reg. No. –141170110007,Class Roll No. –CSE/2014/033)** of B.Tech 8th Semester may be accepted in partial fulfillment for the degree of **Bachelor of Technology in Computer Science & Engineering** under West Bengal University of Technology (WBUT).

.

……

Project Supervisor

Department of Computer Science and Engineering
RCC Institute of Information Technology

Countersigned:

……

Head
Department of Computer Sc. & Engg,
RCC Institute of Information Technology
Kolkata – 700015.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

RCC INSTITUTE OF INFORMATION TECHNOLOGY

CERTIFICATE OF APPROVAL

The foregoing Project is hereby accepted as a credible study of an engineering subject carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein, but approve the project only for the purpose for which it is submitted.

FINAL EXAMINATION FOR
EVALUATION OF PROJECT

1. ——————————————

2. ——————————————

(Signature of Examiners)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

RCC INSTITUTE OF INFORMATION TECHNOLOGY

ACKNOWLEDGEMENT

We express our sincere gratitude to Prof. Sk. Mazharul Islam, Department of CSE, RCCIIT, for extending his valuable time for us to take up this problem as our B.Tech thesis. It was his supervision and guidance that made it possible for us to complete our research. It would have been impossible for us to complete our thesis without his extraordinary support and advice.

_____

_____

_____

_____

Table of Contents

Abstract

In a social network there can be many different kinds of links or edges between the nodes. Those could for example be social contacts, hyper-references or phone-calls. Link Prediction is the problem of predicting edges that either don't yet exist at the given time t or exist, but have not been discovered, are likely to occur in the near future. We develop approaches to link prediction based on measures for analyzing the proximity of nodes in a network. Consider a co-authorship network among scientists, e.g. two scientists who are close in the network will have col-leagues in common, so they are more likely to collaborate in the near future. Our goal is to make this intuitive notion precise and to understand which measures of proximity in a network lead to the most accurate link predictions. Link prediction algorithms can be classified into three categories: Node neighborhood approaches, Path based approaches and Meta approaches. Node neighborhood approach is based on local features of a network, focusing mainly on the nodes structure (i.e. based on the number of common friends

that two users share). The local-based measures are: Common Neighbors, Jaccard's coefficient, Adamic/Adar and Preferential Attachment. Path based algorithms considers the ensemble of all paths between two nodes. The Path based algorithms are: Katz, Sim-Rank, Hitting Time and Commute Time, Rooted PageRank, Prop Flow and High-Performance Link Prediction. Meta-Approaches alter the data before being passed to one of the path-based approaches. The algorithms are: Low-rank approximation, Unseen bigrams and Clustering.

List of Figures

Chapter 1

Introduction

1.1 Introduction to Social Networks

A Social Structure consists of nodes(Individuals or Organizations) and nodes are connected by different types of relationships. A set of social actors or nodes(such as individuals or organizations) and a set of the dyadic ties between these nodes constitute a social network. For example scientists in a discipline, employees in a large company, business leaders can be thought as nodes in a network and co-authors of a paper, working on a project, serve together on board can be thought as edges respectively. The idea behind Social Networks is to create opportunities to develop friendships, share information and promote business in a network. OSN like Facebook and Twitter have become important part of daily life of millions of people. The enormous growth and dynamics of these networks has led to several researches that examine the network properties i.e. structural and behavioral properties of large scale social networks.

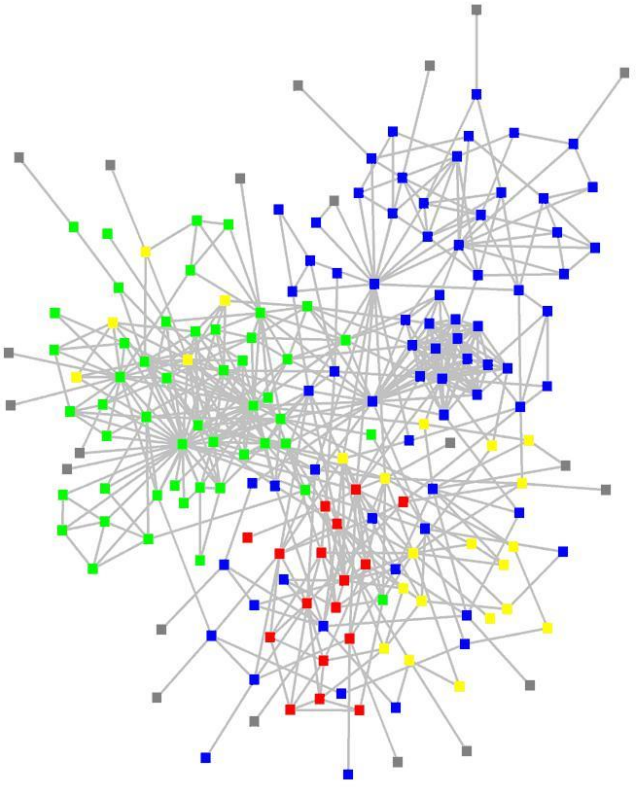


Figure 1.1: Social Network

1.2 Social Network Analysis

Social network analysis(SNA) is in depth analysis of social networks. SNA is the mapping and measuring of relationships, links and owes between nodes(people, groups, organizations, computers) and many other connected entities which pro-

vides some knowledge and information. The vertices or nodes in the network are the people and groups while the links show relationships or owes between the nodes. We can do visual and a mathematical analysis of human relationships through SNA that helps us to make sense out of the social network, to and the complex structure of social networks, to understand the evolution of social networks, network dynamics and to discover complex communication patterns and characteristic features of the network.

1.3 Tasks Of Social Network Analysis

Social networks are dynamic by nature. They change very quickly over a specific interval. Continuously new relationships establish between nodes and many old relationships break. These relational changes(when people become friends through common friends), characteristics of the nodes, characteristics of pairs of actors or link weights and random unexplained events in sequences the graph characteristics. The key tasks of SNA include different measures to rank nodes(or edges), Link prediction problem, Inferring social networks from social events, Viral marketing, Community detection, Design of incentives in networks, Determining implicit social hierarchy, Network formation, Spars cation of social networks(with purpose). There are many measures to rank nodes like degree centrality, closeness central-ity, clustering coe cient, betweenness centrality, Katz centrality and Eigen vector centrality. Link prediction is predicting the links that does not exist or exist, but not known and have probability to occur in the near future. Viral marketing deals with exploiting social connectivity patterns of users to propagate the awareness of product. Community detection involves graph partitioning based on activities over the social network and determining the dense sub graphs in a social network. In designing the incentives, only the person who answers the query is rewarded, with no reward for the intermediaries. Since individuals are often rational and intelligent, they may not participate in answering the queries unless

some kind of incentives are provided. SNA has many applications like information sharing, Information sharing, Understand the spread of diseases, Marketing in e-Commerce and e-Business, determine the in entail entities, build e active social and political campaign, Predict future events, tracking terrorists and location based crowdsourcing.

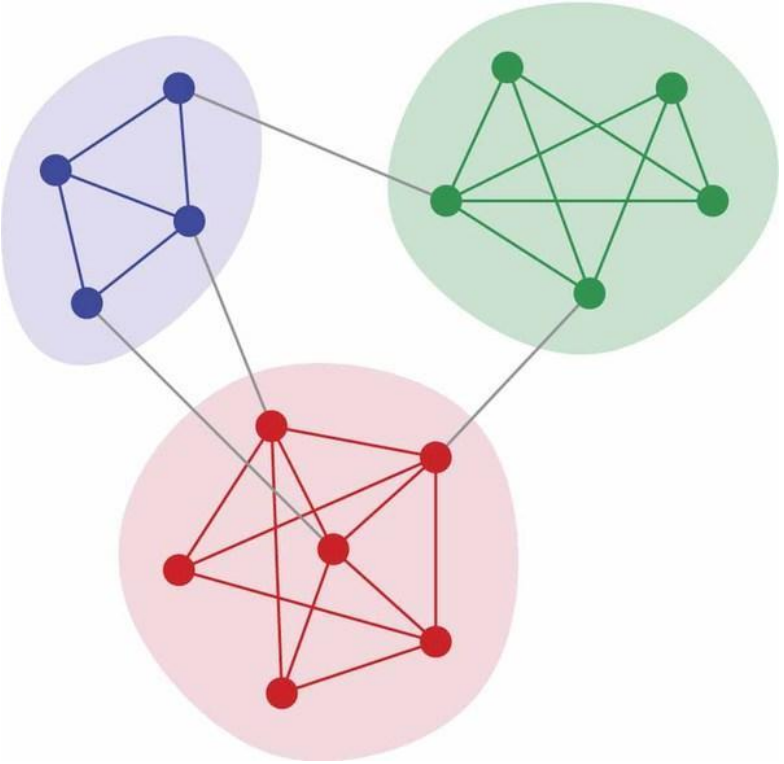


Figure 1.2: Community Detection in Social Networks

1.4 Link Prediction Problem

Different kind of links or edges between the nodes exist in a social network. For example, social contacts, phone-calls or hyper-references. On analysis of social networks, there can be many information about the linkage between the nodes that are not discovered or unknown at a given point of time. Link Prediction is the problem of predicting links that either dont yet exist at the given time t or exist, but unknown up to this time. Given a picture of a social network(nodes and links) at time t, we need to predict accurately the links that will be added to the network during the interval from time t to a given future time t+1. In effect, the link prediction problem concentrates on to what extent can the evolution of a social network be modelled by using intrinsic features of the network itself? Let

us consider a co-authorship network among researchers, for example, there are di erent reasons, outside to the network, why two researchers who have never written a paper together will do so in the next few years. Or, when one of the researchers changes institutions, they may come geographically very close. Such interactions are be hard to predict. But by studying the network characteristics, we can predict the possible links that are going to form. Our objective is to make this intuitive notion very exact, and to understand which measures of proximity in a graph lead to accurate predictions.

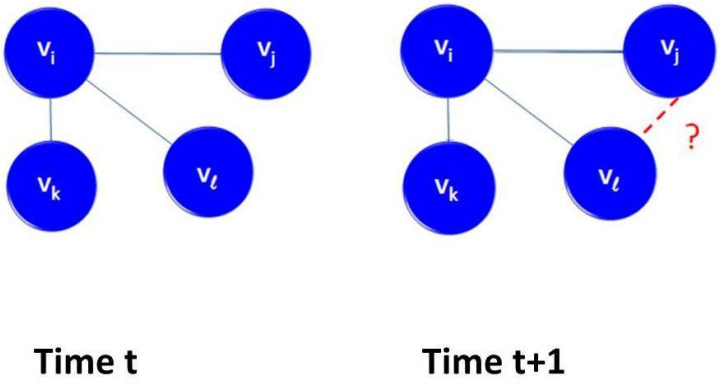


Figure 1.3: Link Prediction in Social Networks

The link prediction problem is also deals with the problem of getting missing links from a known network, in a number of elds. It involves prediction of additional links that are not directly visible currently, are likely to exist in a network based on observable data. It considers a static picture of the network, rather than taking network evolution and network dynamics. It also considers speci c prop-erties of the nodes in the network, rather than computing the power of prediction methods that focuses on the graph structure.

1.5 Application of Link Prediction

Apart from its role as a basic question in social network formation, the link prediction problem could be related to a number of interesting applications of so-cial networks. It is found that a large and medium organisation like a company can bene t from the involvement within the social network among its employees. These bene ts to supplement the organisation hierarchy de ned by the organiza-tion. E ective and e cient methods for link prediction could be used to analyse and study such a social network, and suggest interactions that have not yet been

utilized within the organization, more likely to form. Link Prediction has a great role in security research, largely inspired by the problem of controlling terrorist networks and predicting their future involvement. In bioinformatics, e cient link prediction techniques can be used to predict interactions between proteins. In e-commerce it helps in building the recommendation systems so that helps in viral marketing and e ective product awareness.

Chapter 2

Literature Review

Given a social network G(V; E) in which an edge represents some kind of interactions between its vertices on nodes at a given time t. Suppose we have a snapshot of a social network at a given time. We choose four times $t_0 < t_{00} < t_1 < t_{01}$, and give our algorithm to predict links that are likely to be formed in the near future from the network $G[t_0; t_{00}]$. That results in predicting new links, not present in $G[t_0; t_{00}]$, that are expected to appear in the network $G[t_1; t_{01}]$. We refer to $[t_0; t_{00}]$ as the training interval and $[t_1; t_{01}]$ as the test interval [1].

The most basic approach for similarity between any pair of nodes is by taking the length of their shortest path in graph. We rank pairs of nodes in descending order of score(x; y), where score(x; y) is the negative of the shortest path length between x and y. We take a snapshot of a social network as training set and predict the interactions among the nodes of training set that are likely to occur in near future.The algorithms are classi ed as belows [2].

2.1 NODE NEIGHBORHOOD ALGORITHMS

Node neighborhood meaning the nodes directly connected to the two given nodes. It is simple technique which traverse only paths of length 2. For any node A it check the neighbors of neighbour of A and computes their similarity with A. It considers only local features of a network, focusing mainly on the nodes structure(i.e. based on the number of common friends that two users share).

2.1.1 Common Neighbors

The Common Neighbors method provide a measure for similarity by calculating the intersection of the sets of neighbors of the nodes to predict future linkage. The Common Neighbors(CN) is de ned as follows

$$CN(x; y) := \quad (x) \qquad (y)$$

This measurement is based on the idea that two nodes a and b have an increased probability to connect if they have a shared neighbor c. With a growing number of shared neighbors this probability grows even higher.

The weighted Common Neighbors(CN w) is de ned as follows where w(x; y) is the number of interactions between the nodes x and y.

$$CN\ w(x; y) := \sum_{z\ (x)\setminus(y)} \frac{w(x; z) + w(y; z)}{2}$$

2.1.2 Jaccard coe cient

Jaccards coe cient measures number of the features(neighbors) that are shared between two nodes commensurate to all features that either one of the nodes has. Jaccards coe cient is a normalized variation of Common Neighbors [?]R7). The Jacard coe cient is de ned as follows

$$J(x; y) := \frac{(x) \setminus (y)}{(x) [(y)}$$

This is the Common Neighbors measurement normalized by the union of the node neighborhoods.

2.1.3 Adamic/Adar

It is a measurement that compares how many attributes two nodes have in common. They rate items that are unique to a few users more heavily than items shared amongst a huge group of users. This measurement can easily be adjusted in the context of node neighborhood by looking at shared neighbors as an attribute. Therefore the sum over the shared neighbors inverse of the logarithms of their neighborhoods is proposed [3].
The Adamic/Adar is de ned as follows

$$AA(x; y) := \sum_{z\ \ (x)\setminus(y)} \frac{1}{\log j\ (z)j}$$

The weighted Adamic/Adar (AAw) is de ned as follows where w(x; y) is the number of interactions between the nodes x and y [4].

$$AAw(x; y) := \sum_{z \in (x) \cap (y)} \frac{w(x; z) + w(y; z)}{2} \cdot \frac{1}{\log_{z_0 \in (z)} w(z^0; z)}$$

2.1.4 Preferential Attachment

Preferential Attachment is based on the hypothesis that a node x will get new neighbors faster than a node y given y has less neighbors than x. So the probability that a node will form a new link varies with number of its present neighbors. The likelihood of two nodes being connected by an edge based on preferential attachment is measured by multiplying the number of their neighbors [5]. The Preferential Attachment is de ned as follows

$$P A(x; y) := \quad (x): (y)$$

The weighted Preferential Attachment (P Aw) is de ned as follows where w(x; y) is the number of interactions between the nodes x and y:

$$P Aw(x; y) := \sum_{x; (x)} w(x; x^0): _{y_0 \in (y)} w(y^0; y)$$

2.2 PATH BASED ALGORITHMS

Some measurements of link prediction take all paths between two nodes in consid-eration. The computation of graphs that take the entire graph in consideration is by nature much more complex than node neighborhood algorithms.

2.2.1 Katz

A measurement that takes all paths between two nodes in consideration while rating short paths more heavily. The measurement exponentially reduce the con-tribution of a path to the measure in order to give less weightage longer paths. Therefore it uses a factor of ^{l} where l is the path length.

The Katz is de ned as follows

$$K(x; y) := \sum_{l=1}^{1} {}^{l}{}_{:jpaths^{<l>}{}_{x;yj}}$$

where $paths^{<l>}{}_{x;y}$ the set of all paths from source x to destination y that have the path length l.

Unweighted : $paths^{<l>}{}_{x;y}$ = 1, if x and y have collaborated and 0 otherwise

Weighted : $paths^{<l>}{}_{x;y}$ is the number of times that x and y have collaborated

The can be used to control how much the length of the paths should be considered. A very small concludes to an algorithm where paths of length three or more are taken much less into account and therefor the algorithm converges node neighborhood algorithms. It has roughly cubic complexity as it requires matrix inversion [6].

2.2.2 SimRank

If two nodes are referenced by more similar objects, then the two nodes have large similarity value. Every object obviously has a similarity score of 1 to itself. Node x and node y are then similar to the degree they are joined to similar neighbors [7].
The SimRank is de ned as follows

$$S(x; x) := 1$$

$$S(x; y) := \quad : \quad \frac{\sum_{a\ (x)} \sum_{b\ (y)} S(x; y)}{(x): (y)}$$

is a constant with [0; 1]. The constant can be thought of as a con dence level. If you consider a situation in which a and b are both neighbors to c, than obviously the similarity of c to itself is 1, but we do not want to conclude that s(a; b) = s(c; c) = 1. Instead we let s(a; b) = s(x; x) because we are not as con dent about the similarity of a and b as we are about s(x; x) = 1.

2.2.3 Hitting Time and Commute Time

Starting from a node x a random walk on a given graph moves iteratively over the graph while choosing the next node each step at random. The expected number of steps to get from x to y via a random walk is de ned as the Hitting Time H(x; y). A short hitting time implies node similarity and therefor a heightened chance of future linking. The commute time C(x; y) is a variant of Hitting time which is useful for undirected graphs, because the hitting time is not symmetric. Therefore it is de ned as follows:

$$C(x; y) := H(x; y) + H(y; x)$$

The commute time can have high variance, hence, prediction by this feature can be poor. If z is a node with high stationary probability far o x and y, then a random walker would probably reach the neighborhood of z. To avoid that we can use reset the random walker to x with a xed probability of .

two normalized versions Hitting Time normalized (H_n) and Commute-Time normalized (C_n) are de ned where $_x$ is the stationary probability of x to safeguard it against vertices with a very high :

$$Hn(x; y) := \quad H(x; y): _y$$

$$Cn(x; y) := \quad (H(x; y): _y + H(y; x): _x)$$

2.2.4 Rooted PageRank

Rooted PageRang is a modi cation of the Page Rank measure (which is an attribute of a single vertex) for link prediction. It is the amount of step from x to y with a probability of to return to x each step (and 1 to go to a random neighbor). This metric is asymmetric and can be made symmetric by summing with the counterpart where the role of x and y are reversed [8].

The rooted pagerank(RPR) between all node pairs is calculated as follows: Let D be a diagonal degree matrix de ned as:

$$D[i; i] := \sum_{j} A[i; j]$$

And let N be the following matrix with normalized row sums.

$$N := D^{1}$$

Then the Rooted Pagerank can be calculated as

$$RP R := (1 \quad)(I \quad N)^{1}$$

2.2.5 PropFlow and High-Performance Link Prediction

The unsupervised PropFlow method calculates the probability that a random walker reaches node y from node x in l steps or fewer while using link weights as transition probabilities. If the algorithms revisits any node including x or if it reaches y the algorithm terminates. When compared to Rooted PageRank the algorithm is more localized and is insensitive to topologic noise far from the source node. It is faster to compute because it does not require random resets. High-Performance Link Prediction as a framework for link prediction. They distinguish between two variants:

HPLP: Does not use the existing unsupervised methods, but only simple Measures like In- and Out-Degree, Max. Flow, Shortest Paths or PropFlow

HPLP+: Uses the full feature set adding Adamic/Adar, Jaccards coe cient, Katz and Preferential Attachment [2].

2.2.6 Supervised Random Walks

Node and link attributes along with node structure information are used for prediction. Supervised learning strength is assigned to the edges that are likely to have new links in the future so that random walker can visit them more likely. The Strength is not set manually, but learned from the features of each edge and nodes between them.

2.3 META APPROACHES

Meta-Approaches alter the data before being passed to one of the algorithms mentioned above.

2.3.1 Low-rank approximation

For a lot of the mentioned algorithms there is a equivalent formulation for an adjacency matrix M. For a large Matrix M the Matrix M_k is the rank-k matrix, what can be done e ciently by singular value decomposition [9].

> Katz measure using M_k rather than 4MCommonN eighborsscoringbyinnerproductsof rows rather than M

The contains most related nodes to x under score(x; :) are de ned as $S_x^{< >}$. So after calculating the score(x; y), we need to calculate the $S_x^{< >}$.

$$U\ B(x; y) := jz : z \quad (y) \setminus S_x^{< >} j$$

$$U\ Bw(x; y) := \sum_{z\ (y)\setminus S_x^{< >}} score(x; z)$$

2.3.2 Clustering

This includes improving the quality of the algorithms by a clustering procedure and after that the algorithm is applied to the modi ed sub graph. To achieve that the measure is computing score(x; y) for all edge in the original graph and only keeping the p fraction of these edges, where the score is highest [10]. After that the score algorithm is applied to the modi ed graph. Using this technique, the measurement is only applied to those nodes, in which the scoring algorithm has the most con dence in. This can be seen as a cleaning up by removing of tenuous edges [11].

2.4 Bayesian Probabilistic Model

There are two types of probabilistic approaches to predict links.

The rst approach extends a framework of probabilistic relational models capturing probabilistic interactions between attributes of related entities by modelling interactions between the attributes and the link structure itself [12]. For a probability distribution over a database a template describing the relational schema for the domain and the probabilistic dependencies between attributes of the domain in form of a PRM(probabilistic relational model) is speci ed. Probability distribu-tion on the properties of the nodes and the links can be de ned. By including the links into the probabilistic model they can be used to predict other links and to help make predictions about other attributes in the model. If we look at existence uncertainty no assumptions are made about the number of links that exist they are part of the probabilistic model, but can still be used to make inferences about other attributes in the model [13].

The second approach is based on the topological features of network structures only. A probabilistic evolution model of network structure modelling probabilistic ips of existence of edges depending on a copy-and-paste mechanism of edges is presented. Based on this model a transductive learning algorithm for link pre-diction based on an assumption of the stationarity of the network is proposed. The algorithm realizes a maximum likelihood estimation procedure using expo-nentiated gradient ascent. This is based on the idea that if a node a has a strong in uence on a node b and there is an edge between a and another node c. The authors assume a high probability that a link will establish between b and c and that there is a very low probability that there will never be a node between them [14].

2.5 LINEAR ALGEBRAIC METHOD

It is a general method to solve the link prediction problem which works directly on the graph adjacency matrix or Laplacian matrix. The problem is reduced to a one-dimensional regression problem [9][15, 16]. They training set is reduced to its biggest connected component. The resulting set was then split into two adjacency matrices A and B, where A was the source matrix and B containing one third of

all edges the target matrix. Di erent curve tting methods can be used to predict the edge sets in the test set [17].

Chapter 3

Objective of the Project

The node neighbourhood algorithms of link prediction are based on node proximity of a network. Common neighbours, Adamic/Adar index, Jaccard Coe cient, etc. give the friend suggestion by exploiting path length of 2 between the source and his potential friends. As it only considers of path lengths of 2, so it cant give e cient prediction because it only focuses on immediate neighbours of the node. But many other aspects are also needed to predict e ciently. Assuming a node can be connected to others by di erent paths and of di erent path lengths, so two nodes which are connected with many unique ways, are more likely to be connected and that varies with the path lengths of di erent ways in which they are connected. Our Proposed algorithm is more e cient than the algorithms which considers all the paths of the network. Because considerideration of all the paths require more time and space complexity. Sometimes it is necessary to get good result with in less interval of time.

Our proposed algorithm performs better than global approaches as it is based on user input bounded path traversal. It predicts the links by traversing to a certain path length given by the user. In global approaches, the time and space complexity are high due to consideration of all paths in the network. As our algorithm only traverse up to certain path length, so its complexity is low with comparison to global approaches. It also outperforms the node neighbourhood algorithms as it traverses more path lengths than the node neighbourhood algorithms. It considers the network characteristics around the target node to predict its future links.

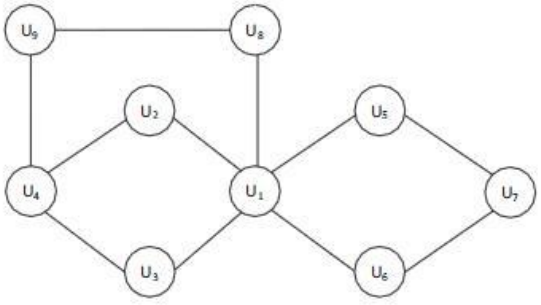


Figure 3.1: Link PredictionProblem

When we considers only path lengths of 2, then U_4 and U_7 have the equal probability of forming links with U_1 as they are connected with two di erent way of path length 2. But when we consider the path length of 3, then U_4 has more

probability to form a link than U_7. Because U_4 is connect with three di erent

paths with U_1. If we have followed node neighbourhood techniques, then we will get equal probability of U_4 and U_7 getting connected with U_1. But through our algorithm with max traversal length of 3, we can conclude that U_4 has more chance to get connected with U_1 than U_7.

So our algorithm performs better than node neighborhood algorithms in terms of efficiency of result. It also performs better than global approaches in terms of time and space complexity.

Chapter 5

Methodology for implementation

The link prediction algorithms based on user input as maximum path to be traversed predicts the probability of formation of link between any two nodes of the network by traversing all the paths of the network up to that certain input path length. It rest traverses the path lengths of 2 i.e. the immediate neighborhoods of the node. we can say it runs a neighborhood algorithm on path length 2. It then produces a similarity list between every two nodes. When it traverses the graph for path length 3, it uses new paths that are made by path length 2 to get the new path lengths of 3 and computes their similarity matrix and updates the similarity matrix in a cumulative way. This process continues till the graph is traversed up to the maximum path length to be traversed given by user. Let we get a path from A to B while traversing for the path length of n 1 with some similarity value, when we traverse the graph for path length value n, then we will check all the neighbors of B(i.e. C) to get path from A to C. We compute the similarity of each pair (A; C) and update the path list if their no direct link between A and C in the original graph.

The inputs to the algorithm are the graph in terms of a list or adjacency array, the total number of nodes the graph, maximum length to be traversed, which determines how many time the algorithm will run and the path length for each speci c traversal. The output of the algorithm is a similarity list containing the similarity value between every two nodes by traversing the path lengths of given maximum user input value. By observing the similarity matrix we can predict the future links. The high similar values have more probability to form links in near future and we can classify the values based on certain threshold value. The similarity values more than the threshold value are likely to form future links. This prediction can be compared with the test data to get the e ciency of the algorithm.

5.1 The Proposed Algorithm

For each path length, we have to follow many steps:

Calculate that path list with respect to the list with previous input path value

Update the current adjacency list for the entries having non-zero path value

Calculate the similarity measure with respect to the corresponding path list Update the similarity list by adding the new similarity measures to the list Increment the path length

This iteration stops when current path length exceeds the maximum value of path length to be traversed.

5.1.1 Algorithm Parameters

5.1.1.1 Input Parameters

A : adjacency matrix of undirected and unweighted graph

n : total number of nodes of the graph

l : max length of path to be explored in G

m : the length of a path for current iteration

5.1.1.2 Output Parameters

sim(i; j) : Similarity measure between nodes i and j

5.1.2 Algorithm

Algorithm 1: MAIN FUNCTION

```
for m      2 to n do
    cpath(A,n,prev,or) sim simi(sim,path,n,m);
    path 0 ;
end
```

The main Program iteratively calls for checking the new collaboration between any two nodes for a speci c path length through cpath function and computes the

similarity between the new collaborations with exactly m path length and updates the similarity measures through sim function.

```
for i      1 to N do
    for j   1 to N do
        if i < j then
            if or (i,j) 6= j then
                for k   1 to N do
                    if  prev (i,k) 6= 0 then
                        if  A (i,k) 6= 0 and or (k,j) 6= 0 then
                            path (i,j)   path (i,j)+prev (i,k) * or (k,j) ;
                        end
                    end
                end
                path (j,i)     path (i,j) ;
            end
        end
    end
end
prev    path;
for i   1 to N do
    for j   1 to N do
        if path (i,j) 6= 0   then
            A (j,i)     1 ;
        end
    end
end
return path and A;
```

The cpath function rst checks whether there is path from any two nodes of m path length. It checks it by merging the new paths generated while traversing the previous path length (m 1) and their neighbors in the original graph. So path

matrix contains new collaborations of path length exactly m.

Algorithm 2: FUNCTION SIMI

```
for i      1 to n do
    for j   1 to n do
        lower    1;
        for k   1 to m do
            lower      lower * (n-k) ;
        end
        sim (i,j)      ( 1/(m-1)* sim (i,j) ) / lower;
    end
end
return sim;
```

The simi function nds the similarity measure for every pair of nodes which have path length exactly m. It then cumulates the similarity values till the path length of l for every two nodes. The pair of nodes having higher value of similarity are more likely to form link in the near future.

Chapter 6

Implementation Details

A social network contains nodes and edges represent collaboration between nodes. Suppose we have a snapshot of a social network at a given time. We record multiple interactions between every pair of nodes in di erent time-stamps. We choose four times $t_0 < t_{00} < t_1 < t_{01}$, and give our algorithm to predict links that are likely to be formed in the near future from the network $G[t_0; t_{00}]$. That results in predicting new links, not present in $G[t_0; t_{00}]$, that are expected to appear in the network $G[t_1; t_{01}]$. We refer to$[t_0; t_{00}]$ as the training interval and $[t_1; t_{01}]$ as the test interval.

As social networks grow very rapidly and exponentially, So there may be may nodes that may not be present in our snapshot of network on which we are pre-dicting. So we will consider only the nodes that are present in our network and their collaborations will be studied. Thus, in evaluating link prediction methods, we will generally use two parameters training set $G[t_0; t_{00}]$ and test set $G[t_1; t_{01}]$. We will then evaluate how accurately the new edges between elements of training set can be predicted.

6.1 Data Sources

6.1.1 Condmat

The rst data source is a stream of 19,464 multi-agent events representing con-densed matter physics collaborations from 1995 to 2000. We construct weighted, undirected networks from the collaborations by creating a node for each author in the event and a weighted, undirected link connecting each pair of authors. Weights correspond to the number of collaborations two authors share. We use the years 1995 to 1999 (13.9K nodes, 80.6K links) for extracting features as training set and the year 2000 (8.5K nodes,41.0K links) for obtaining ground truth.

6.1.2 Disease-g

The disease-gene(disease-g) network was constructed from three individual data sets. As the name suggests, this network has two distinct node types, diseases and genes, with four edge types connecting them. It also contains genetic associations, protein - protein interactions, phenotypic links and family links as edges. Here we

have taken a small part of it. The nodes are the diseases and the weights represent the genetic similarity between cancer diseases.

6.2 Network Characteristics

	Condmat	Disease-g
Nodes	17636	1835
Edges	23709	7817
Assortativity Coe cient	0.177	0.31
Avg. Clustering Coe cient	0.642	0.665
Number of SSCs	652	1
Largest SSC	15,081	399
Largest SSC diameter	19	4

6.3 Codes

```
import numpy as np
import os
import glob
import random
from random import shuffle
from random import seed
import matplotlib.pyplot as plt
import time
import datetime
import collections
import csv

k = 3           # Top k recomendations for a target user
maxl = 2        # Number of iterations for Katz Algorithm
beta = 0.1      # The damping factor for Katz Algorithm


##################################
######## Helper Functions #########
##################################

# load edge-list from file
def get_edge_list(dataset_path):
        data_file = open(dataset_path)
        edge_list = map(lambda x:tuple(map(int,x.split())),data_file.read().split("\n")[:-1])
        data_file.close()
        return edge_list



# Get the similarity product for a path
# (product of path-step similarities)
def get_sim_product(sim, shortest_path):
        prod = 1
        for i in range(len(shortest_path) - 1):
                prod *= sim[shortest_path[i]][shortest_path[i+2]]
        return round(prod,3)




# Filter out, Sort and Get top-K predictions
def get_top_k_recommendations(graph,sim,i,k):
        return  sorted(filter(lambda x: i!=x and graph[i,x] != 1,range(len(sim[i]))) ,
key=lambda x: sim[i][x],reverse=True)[0:k]
```

```
# Convert edge_list into a set of constituent edges
def get_vertices_set(edge_list):
	res = set()
	for x,y in edge_list:
		res.add(x)
		res.add(y)
	return res




# Split the dataset into two parts (50-50 split)
# Create 2 graphs, 1 used for training and the other for testing
def split_data(edge_list):
	random.seed(350)
	indexes = range(len(edge_list))
	test_indexes = set(random.sample(indexes, len(indexes)/2)) # removing 50% edges
from test data
	train_indexes = set(indexes).difference(test_indexes)
	test_list = [edge_list[i] for i in test_indexes]
	train_list = [edge_list[i] for i in train_indexes]
	csv_file = open('test.csv',"w")
	fields = ['Node1','Node2']
	thewriter = csv.DictWriter(csv_file,fieldnames=fields)
	for i in range(len(test_list)):
		thewriter.writerow({'Node1' : str(edge_list[i][0]) , 'Node2' :
str(edge_list[i][1])})
	return train_list,test_list




# Calculates accuracy metrics (Precision & Recall),
# for a given similarity-model against a test-graph.
def
print_precision_and_recall(sim,train_graph,test_graph,test_vertices_set,train_vertices_set,es
im):
	precision = recall = c = 0
	for i in test_vertices_set:
		if i in train_vertices_set:
			actual_friends_of_i = set(test_graph.neighbors(i))

			# Handles case where test-data < k
			if len(actual_friends_of_i) < k:
				k2 = len(actual_friends_of_i)
			else:
				k2 = k

			top_k = set(get_top_k_recommendations(train_graph,esim,i,k2))

			precision += len(top_k.intersection(actual_friends_of_i))/float(k2)
```

```
                        recall +=
len(top_k.intersection(actual_friends_of_i))/float(len(actual_friends_of_i))
                        c += 1
        #print(esim)
        print "Precision is : " + str(precision/c)
        print "Recall is : " + str(recall/c)




def get_recomemendations(edge_list,esim,name):
        graph = Graph(edge_list)
        edge_vertices_set = get_vertices_set(edge_list)

        #output to a file name output.txt
        if name == "train":
                csv_file = open('train_recommendations.csv',"w")
                fields = ['Node','R1','R2','R3']
                thewriter = csv.DictWriter(csv_file,fieldnames=fields)
                csv_file2 = open('train_edgelist.csv',"w")
                fields2 = ['Node1','Node2','Weight']
                thewriter2 = csv.DictWriter(csv_file2,fieldnames=fields2)
        if name == "total":
                csv_file = open('total_recommendations.csv',"w")
                fields = ['Node','R1','R2','R3']
                thewriter = csv.DictWriter(csv_file,fieldnames=fields)
                csv_file2 = open('total_edgelist.csv',"w")
                fields2 = ['Node1','Node2','Weight']
                thewriter2 = csv.DictWriter(csv_file2,fieldnames=fields2)


        for i in edge_vertices_set:
                if i in edge_vertices_set:
                        actual_friends_of_i = set(graph.neighbors(i))

                        # Handles case where test-data < k
                        if len(actual_friends_of_i) < k:
                                k2 = len(actual_friends_of_i)
                        else:
                                k2 = k

                        top_k = get_top_k_recommendations(graph,esim,i,k2)
                        if len(top_k) == 3:
                                thewriter.writerow({'Node' : str(i) , 'R1' : top_k[0] , 'R2' :
top_k[1] ,'R3' : top_k[2]}) #write to csv file
                        elif len(top_k) == 2:
                                thewriter.writerow({'Node' : str(i) , 'R1' : top_k[0] , 'R2' :
top_k[1]}) #write to csv file
                        elif len(top_k) == 1:
                                thewriter.writerow({'Node' : str(i) , 'R1' : top_k[0]}) #write to
csv file
        i=0
        for i in range(len(edge_list)):
```

```
                thewriter2.writerow({'Node1' : str(edge_list[i][0]) , 'Node2' :
str(edge_list[i][1]) ,'Weight' : str(esim[edge_list[i][0]][edge_list[i][1]])})



# http://be.amazd.com/link-prediction/
def similarity(graph, i, j, method):
        if method == "common_neighbors":
                return len(set(graph.neighbors(i)).intersection(set(graph.neighbors(j))))
        elif method == "jaccard":
                return
len(set(graph.neighbors(i)).intersection(set(graph.neighbors(j))))/float(len(set(graph.neighbo
rs(i)).union(set(graph.neighbors(j)))))
        elif method == "adamic_adar":
                return sum([1.0/math.log(graph.degree(v)) for v in
set(graph.neighbors(i)).intersection(set(graph.neighbors(j)))])
        elif method == "preferential_attachment":
                return graph.degree(i) * graph.degree(j)
        elif method == "friendtns":
                return round((1.0/(graph.degree(i) + graph.degree(j) - 1.0)),3)


###################################
### Methods for Link Prediction ###
###################################

def local_methods(edge_list,method):

        graph = Graph(edge_list)
        edge_n = graph.vcount()
        edge_vertices_set = get_vertices_set(edge_list)



        train_list, test_list = split_data(edge_list)
        train_graph = Graph(train_list)
        test_graph = Graph(test_list)
        train_n =  train_graph.vcount() # This is maximum of the vertex id + 1
        train_vertices_set = get_vertices_set(train_list) # Need this because we have to only
consider target users who are present in this train_vertices_set
        test_vertices_set = get_vertices_set(test_list) # Set of target users

        sim = [[0 for i in range(train_n)] for j in range(train_n)]
        for i in range(train_n):
                for j in range(train_n):
                        if i!=j and i in train_vertices_set and j in train_vertices_set:
                                sim[i][j] = similarity(train_graph,i,j,method)



        sp1 = {}
        for i in train_vertices_set:
                sp1[i] = train_graph.get_shortest_paths(i)
```

```
	# Extended Sim matrix for train_list
	esim1 = [[0 for i in range(train_n)] for j in range(train_n)]
	for i in range(train_n):
		for j in range(train_n):
			if i!=j and i in train_vertices_set and j in train_vertices_set:
				if len(sp1[i][j]) == 0: # no path exists
					esim1[i][j] = 0
				elif train_graph[i,j] == 1 and train_graph[j,i] == 1: # are
neighbors
					esim1[i][j] = sim[i][j]
				else:
					esim1[i][j] = get_sim_product(sim,sp1[i][j])
			elif i == j and i in train_vertices_set and j in train_vertices_set:
				esim1[i][j] = 1
	get_recomemendations(train_list,esim1,'train')



	sim1 = [[0 for i in range(edge_n)] for j in range(edge_n)]
	for i in range(edge_n):
		for j in range(edge_n):
			if i!=j and i in edge_vertices_set and j in edge_vertices_set:
				sim1[i][j] = similarity(graph,i,j,method)
			elif i == j and i in edge_vertices_set and j in edge_vertices_set:
				sim1[i][j] = 1



	sp = {}
	for i in edge_vertices_set:
		sp[i] = graph.get_shortest_paths(i)

	# Extended Sim matrix for total_graph
	esim = [[0 for i in range(edge_n)] for j in range(edge_n)]
	for i in range(edge_n):
		for j in range(edge_n):
			if i!=j and i in edge_vertices_set and j in edge_vertices_set:
				if len(sp[i][j]) == 0: # no path exists
					esim[i][j] = 0
				elif graph[i,j] == 1 and graph[j,i] == 1: # are neighbors
					esim[i][j] = sim1[i][j]
				else:
					esim[i][j] = get_sim_product(sim1,sp[i][j])
			elif i == j and i in edge_vertices_set and j in edge_vertices_set:
				esim[i][j] = 1
	get_recomemendations(edge_list,esim,'total')

	print_precision_and_recall(sim,train_graph,test_graph,test_vertices_set,train_vertices
_set,esim1)
```

```
# Calculates the Katz Similarity measure for a node pair (i,j)
def katz_similarity(katzDict,i,j):
	l = 1
	neighbors = katzDict[i]
	score = 0

	while l <= maxl:
		numberOfPaths = neighbors.count(j)
		if numberOfPaths > 0:
			score += (beta**l)*numberOfPaths

		neighborsForNextLoop = []
		for k in neighbors:
			neighborsForNextLoop += katzDict[k]
		neighbors = neighborsForNextLoop
		l += 1

	return score




# Implementation of the Katz algorithm
def katz(edge_list,method):
	train_list, test_list = split_data(edge_list)
	train_graph = Graph(train_list)
	test_graph = Graph(test_list)
	train_n = train_graph.vcount()
	train_vertices_set = get_vertices_set(train_list) # Need this because we have to only
consider target users who are present in this train_vertices_set
	test_vertices_set = get_vertices_set(test_list) # Set of target users

	# build a special dict that is like an adjacency list
	katzDict = {}
	adjList = train_graph.get_adjlist()

	for i, l in enumerate(adjList):
		katzDict[i] = l

	sim = [[0 for i in xrange(train_n)] for j in xrange(train_n)]
	for i in xrange(train_n):
		if i not in train_vertices_set:
			continue

		for j in xrange(i+1, train_n):
			if j in train_vertices_set:		# TODO: check if we need this
				sim[i][j] = sim[j][i] = katz_similarity(katzDict,i,j)
```

```
	print_precision_and_recall(sim,train_graph,test_graph,test_vertices_set,train_vertices
_set,sim)




# Implementation of the friendTNS algorithm
def friendtns(edge_list, method):

	graph = Graph(edge_list)
	edge_n = graph.vcount()
	edge_vertices_set = get_vertices_set(edge_list)

	train_list, test_list = split_data(edge_list)
	train_graph = Graph(train_list)
	test_graph = Graph(test_list)
	train_n =  train_graph.vcount() # This is maximum of the vertex id + 1
	train_vertices_set = get_vertices_set(train_list) # Need this because we have to only
consider target users who are present in this train_vertices_set
	test_vertices_set = get_vertices_set(test_list) # Set of target users

	sim = [[0 for i in range(train_n)] for j in range(train_n)]
	for i in range(train_n):
		for j in range(train_n):
			if i!=j and i in train_vertices_set and j in train_vertices_set and
train_graph[i,j] != 0:
				sim[i][j] = similarity(train_graph,i,j,method)


	# Calculate Shortest Paths from each vertex to every other vertex in the
train_vertices_set
	sp = {}
	for i in train_vertices_set:
		sp[i] = train_graph.get_shortest_paths(i)

	# Extended Sim matrix
	esim = [[0 for i in range(train_n)] for j in range(train_n)]
	for i in range(train_n):
		for j in range(train_n):
			if i!=j and i in train_vertices_set and j in train_vertices_set:
				if len(sp[i][j]) == 0: # no path exists
					esim[i][j] = 0
				elif train_graph[i,j] == 1 and train_graph[j,i] == 1: # are
neighbors
					esim[i][j] = sim[i][j]
				else:
					esim[i][j] = get_sim_product(sim,sp[i][j])
	get_recomemendations(train_list,esim,'train')
```

```
	sim1 = [[0 for i in range(edge_n)] for j in range(edge_n)]
	for i in range(edge_n):
		for j in range(edge_n):
			if i!=j and i in edge_vertices_set and j in edge_vertices_set:
				sim1[i][j] = similarity(graph,i,j,method)
			elif i == j and i in edge_vertices_set and j in edge_vertices_set:
				sim1[i][j] = 1

	sp1 = {}
	for i in edge_vertices_set:
		sp1[i] = graph.get_shortest_paths(i)

	# Extended Sim matrix
	esim1 = [[0 for i in range(edge_n)] for j in range(edge_n)]
	for i in range(edge_n):
		for j in range(edge_n):
			if i!=j and i in edge_vertices_set and j in edge_vertices_set:
				if len(sp1[i][j]) == 0: # no path exists
					esim1[i][j] = 0
				elif graph[i,j] == 1 and graph[j,i] == 1: # are neighbors
					esim1[i][j] = sim1[i][j]
				else:
					esim1[i][j] = get_sim_product(sim1,sp1[i][j])
			elif i == j and i in edge_vertices_set and j in edge_vertices_set:
				esim1[i][j] = 1
	get_recomemendations(edge_list,esim1,'total')
	print_precision_and_recall(sim,train_graph,test_graph,test_vertices_set,train_vertices
_set,esim)




#####################################
############# Main ################
#####################################

def main():
	# default-case/ help
	if len(sys.argv) < 3 :
		print "python link_prediction.py
<common_neighbors/jaccard/adamic_adar/preferential_attachment/katz/friendtns>
data_file_path"
		exit(1)

	# Command line argument parsing
	method = sys.argv[1].strip()
	dataset_path = sys.argv[2].strip()
	edge_list = get_edge_list(dataset_path)
```

```
	if method == "common_neighbors" or method == "jaccard" or method ==
"adamic_adar" or method == "preferential_attachment":
		local_methods(edge_list,method)
	elif method == "katz":
		katz(edge_list,method)
	elif method == "friendtns":
		friendtns(edge_list,method)
	else:
		print "python link_prediction.py
<common_neighbors/jaccard/adamic_adar/preferential_attachment/katz/friendtns>
data_file_path"

if __name__ == "__main__":
	main()
```

Chapter 7

Results and Analysis

7.1 The Evaluation Measures

Let us de ne P as positive result and N as negative result. The possible outcomes can be formulated in a matrix.

	actual value		
	p	*n*	total

Figure 6.1: Confusion matrx

7.1.1 Receiver Operating Characteristic(ROC)

ROC is the variation of true positive rate(TPR) with respect to false positive rate (FPR) at various threshold settings. So it is de ned by FPR and TPR as x and y-axis respectively.

TPR = fraction of true positives out of total positives i.e. T P=P = T P=(T P +F N)

FPR = fraction of false positives out of total negatives i.e. F P=N = F P=(F P + T N)

7.1.2 Precision-Recall Curve

ROC is the variation of Precision with respect to Recall at various threshold settings. So, it is designed by Recall and Precision as x and y-axis respectively. Precision = Positive predictive value i.e. T P=(T P + F P)

Recall = fraction of true positives out of total positives i.e. T P=P = T P=(T P + F N)

7.2 Comparison between various link Prediction algorithms

There are mainly two types of link prediction algorithms-node neighborhood techniques and global approaches. We have applied different algorithms to Condmat data set.

7.2.1 ROC curve for Condmat data set

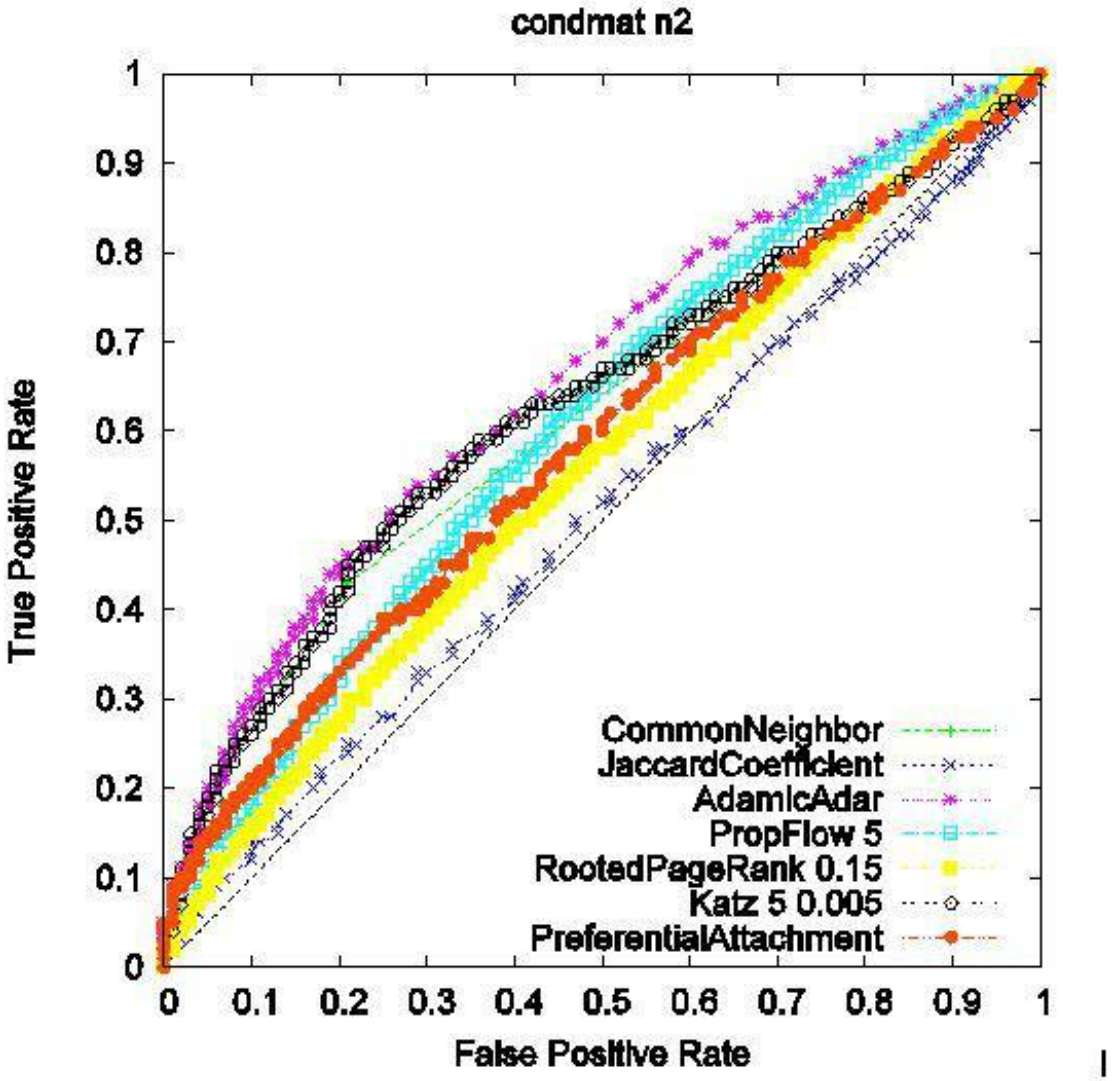


Figure 6.2: ROC curve of condmat data

Here we find that Adamic/Adar outperforms the other algorithms.

7.2.2 Precision Recall Curve for Condmat data set

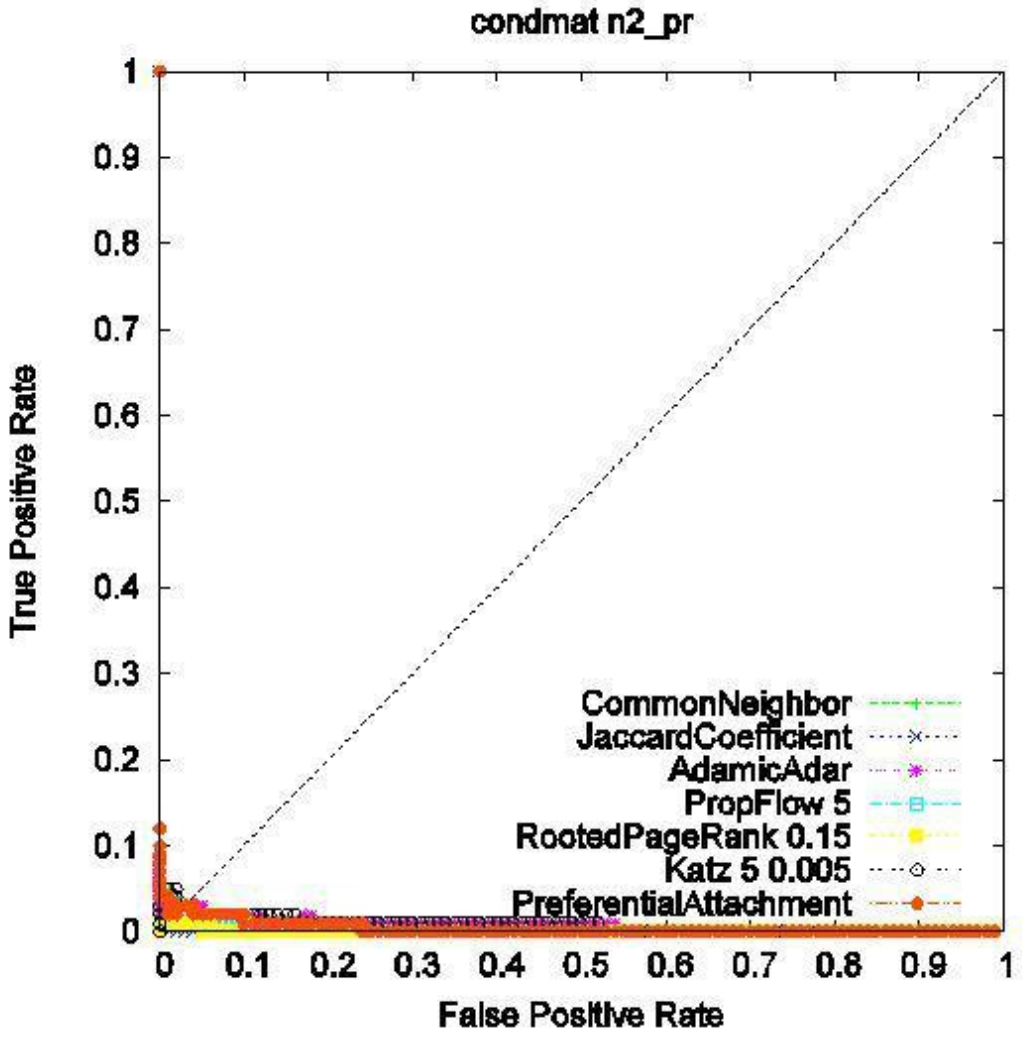


Figure 6.3: PR curve of condmat data

Here also we observe that Adamic/Adar outperforms the other algorithms.

7.2.3 ROC curve for Disease-g data set

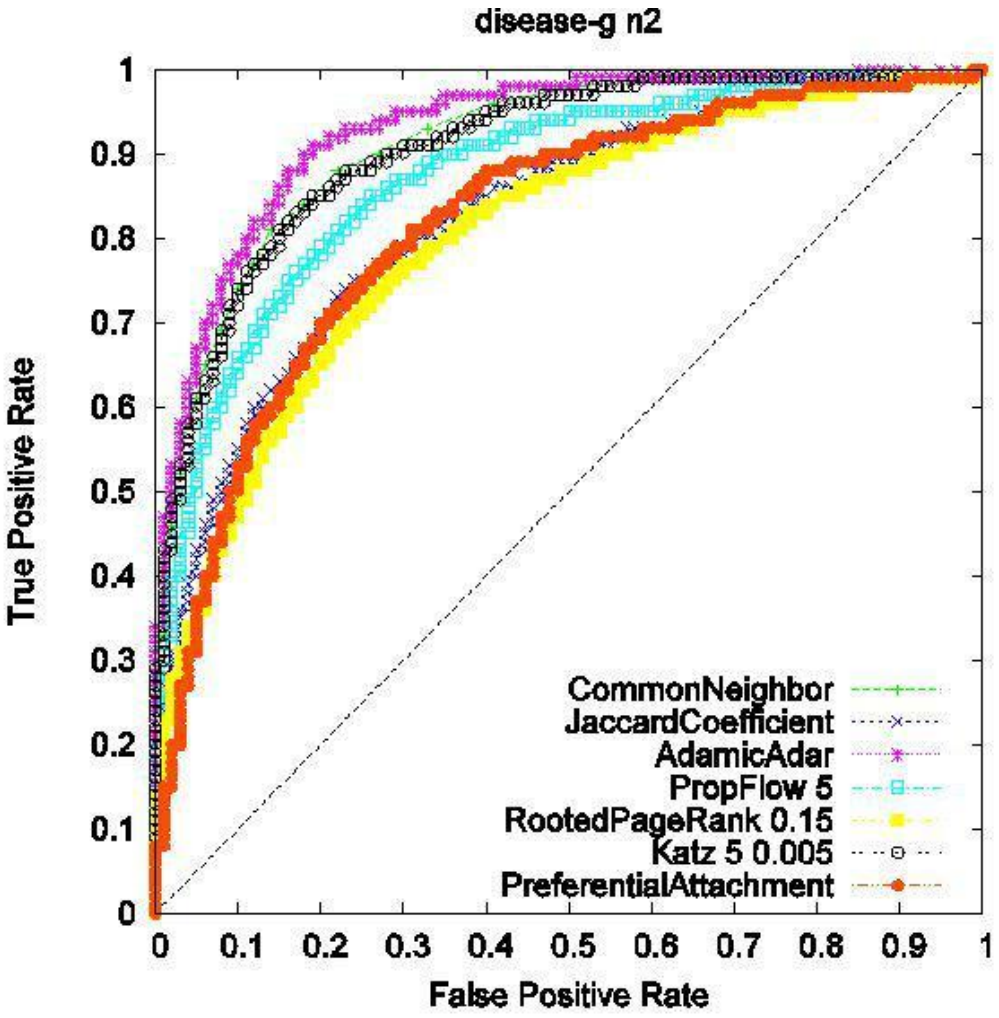


Figure 6.4: ROC curve of Disease-g data

Here we nd that the result remains the same. Adamic/Adar performs well with comparison to the other algorithms.

7.2.4 Precision Recall Curve for Disease-g data set

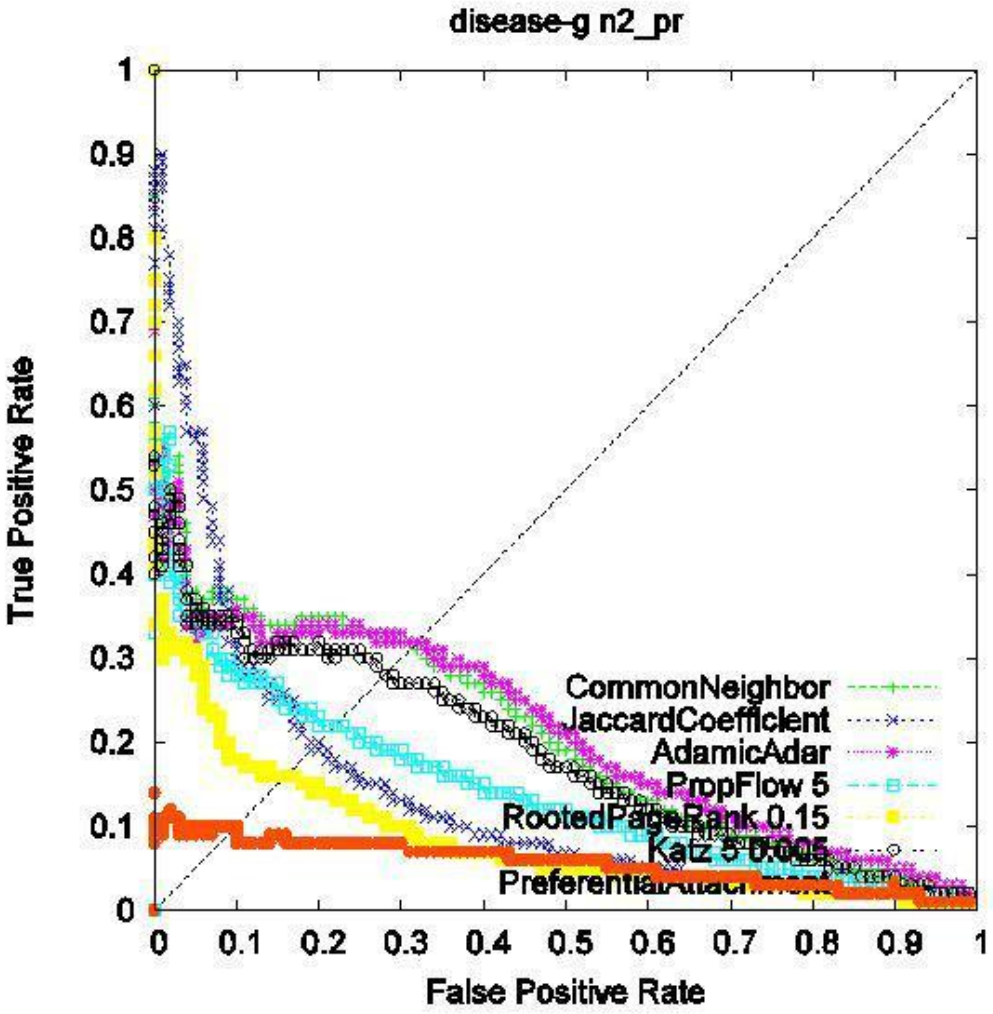


Figure 6.5: PR curve of Disease-g data

Adamic/Adar is the best and most stable graph proximity measurement of the unweighted node neighbourhood based algorithms in almost all categorieswhen it doesnt score highest, the di erence is not much.

7.3 Our Proposed Method Implementation

We are implementing our proposed method on a small network. The input to the algorithms is the network below and the output is a similarity matrix. It has been discussed that with path length of 3, U_4 has more probability to form link with U_1 with comparison to U_7.

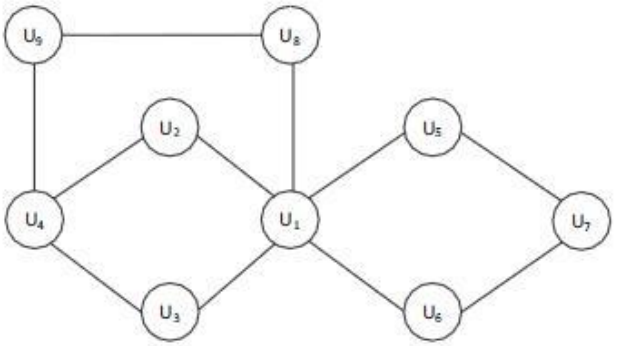


Figure 6.6: Link Prediction Analysis

The algorithm is implemented using maximum length to be traversed as 4. This result shows that the similarity value of the node pair U_1 and U_4 is maximum

```
sim =

         0         0         0    0.3013         0         0    0.2857         0    0.1690
         0         0    0.2857         0    0.1466    0.1466    0.0253    0.1568    0.1568
         0    0.2857         0         0    0.1466    0.1466    0.0253    0.1568    0.1568
    0.3013         0         0         0    0.0281    0.0281    0.0089    0.1690         0
         0    0.1466    0.1466    0.0281         0    0.2857         0    0.1456    0.0171
         0    0.1466    0.1466    0.0281    0.2857         0         0    0.1456    0.0171
    0.2857    0.0253    0.0253    0.0089         0         0         0    0.0253    0.0055
         0    0.1568    0.1568    0.1690    0.1456    0.1456    0.0253         0         0
    0.1690    0.1568    0.1568         0    0.0171    0.0171    0.0055         0         0

>>
```

Figure 6.7: Similarity matrx

and more than that of the pair U_1 and U_7. So, there is more probability of formation of link between U_1 and U_4. Hence our proposed method works correctly.

7.4 Complexity Analysis

Global approaches traverses all paths of the network to predict the links. They require matrix inversion. So the time complexity for global techniques is $O(n^3)$.

The node neighborhood approaches traverses path length of 2 in a network. That means, for any node it rst traverses all its neighbors and then their neighbors and computes the similarity of the source node with its neighbors neighbor. Let h be the average node degree of the network. As h << n, so the time complexity is $(n*h^2)$. As our method is based on bounded length traversal, it traverses up to a path length of l. So the time complexity is $O(n*h^l)$ and the space complexity is O(n*h).So Our method is better than global approaches in terms of complexity.

	Condmat	Disease-g
Page Rank	370 sec	530 sec
katz	430 sec	620 sec
Node neighbor	72 sec	230 sec
Proposed	260 sec	350 sec

This result shows that our algorithm takes less time as compared to the global approaches. Node neighbor approach is taking less time as it focuses on only immediate surroundings of the node and traverse only paths of length 2.

The below gure shows the bar plot of the time taken by di erent algorithms and the proposed algorithms.

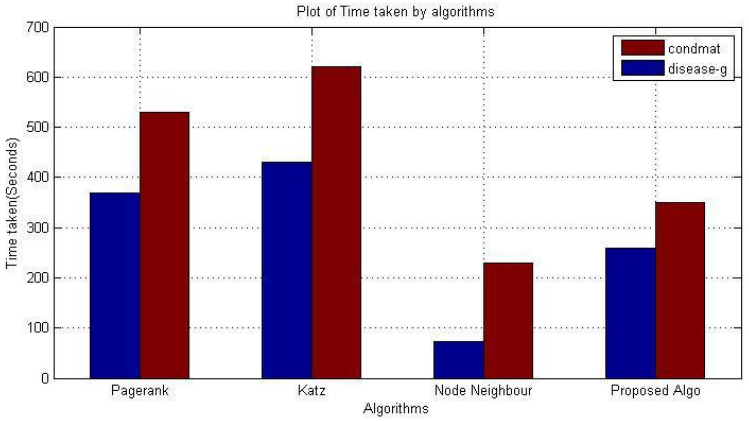


Figure 6.8: Bar Plot of Time takes by algorithms

Chapter 8

Conclusions and Future Works

8.1 Conclusions

Link Prediction is the method to predict the possible future interactions among the nodes in the near future. Our algorithm uses both global and local characteristics of the network to predict the links. Global approaches has the time constraint as they traverse all paths of network to predict the links and local approaches are less e cient as they consider only local features of the node. Our approach is compared with all the approaches and it provides e cient and accurate friend suggestions in a less interval of time

8.2 Future Research Opportunities

Link Prediction based on other features like photo, video tagging can be used for better prediction. As many features as we consider simultaneously, the prediction will be better because it gives information about many ways peoples may by connected. We can consider the positive as well as negative links in a network. If positive weight is for support, then negative weight should be for opposing it. As network is always dynamic, so we can consider network dynamics into consideration.