Learning Models for Influence Maximization

Thesis by

Satyaki Sikdar 12600113093
Dyuti De 12600113045

In Partial Fulfillment of the Requirements

for the Degree of

Bachelor of Technology

in

Computer Science and Engineering

Heritage Institute of Technology, Kolkata

Maulana Abul Kalam Azad University of Technology, West Bengal

2017

ACKNOWLEDGEMENT

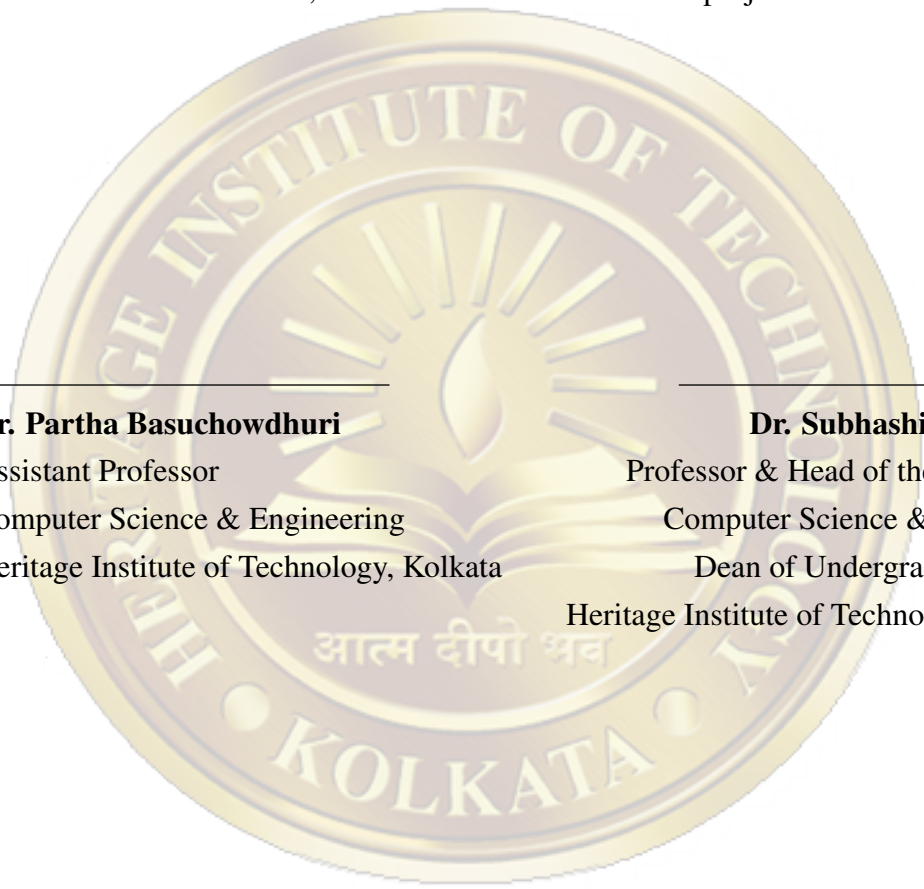
First of all, we would like to express profound gratitude to our guides, *Dr. Partha Basuchowdhuri* and *Dr. Subhashis Majumder* for their unending support, guidance and supervision throughout the duration of the project.

We also extend our gratitude towards the entire department of Computer Science and Engineering at Heritage Institute of Technology, Kolkata for their continuous support. We are grateful for the easy access to the state-of-the-art server of the department as much of the experiments demanded access to heavy computational power.

Satyaki Sikdar **Dyuti De**

CERTIFICATE

Certified that the project report titled *Leaning Models for Influence Maximisation* is the bonafide work of *Satyaki Sikdar*, B.Tech. 4^{th} Year, Roll Number - 12600113093, Registration Number - 131260110093, and *Dyuti De*, B.Tech. 4^{th} Year, Roll Number - 12600113045, Registration Number - 131260110045, both of whom carried out the project under our supervision.



Dr. Partha Basuchowdhuri
Assistant Professor
Computer Science & Engineering
Heritage Institute of Technology, Kolkata

Dr. Subhashis Majumder
Professor & Head of the Department
Computer Science & Engineering
Dean of Undergraduate Studies
Heritage Institute of Technology, Kolkata

External Examiner

Contents

List of Figures

List of Algorithms

Abstract

Information diffusion through social networks though hard to predict and analyze, plays a pivotal role in our everyday decision making. Whether it is through word of mouth or clever advertising strategies, we live in an illusion of free will. Whatever the medium, it's the underlying interaction networks that govern the spread of information and ultimately our actions. In this work, we first lay the groundwork for modeling the flow of information based on the Independent Cascade(IC) model and PageRank, before proceeding to establish an experimental setup allowing us to observe the flow of information in a real life network — a very popular restaurant search and discovery service — underlying which there is an information rich bipartite network of users and restaurants. A recommender system is made using this bipartite network which performs exceedingly well when compared to regular collaborative filtering.

Keywords: information diffusion, influence maximization, independent cascade, bipartite networks, clustering, recommender system, collaborative filtering

Chapter 1

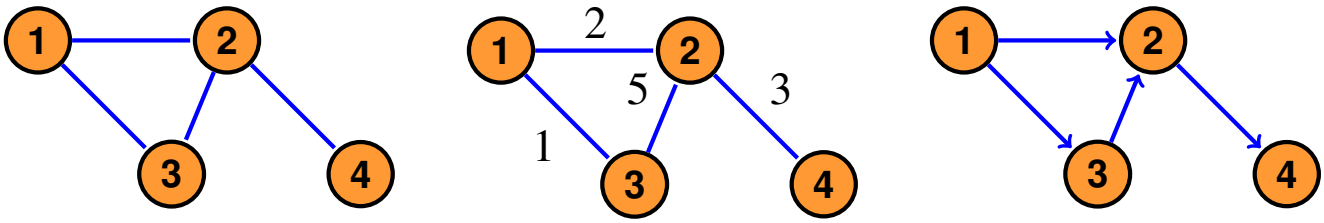
Introduction

The world we live in is, and has always been fascinating. For a long time, man struggled to make sense of the events happening around him, but that didn't stop him from reasoning about them. As time went on, the reasoning became more logically sound, but largely due to the absence of a grand unifying theory, there still isn't an one true answer, explaining everything there was, is, and ever will be.

There exists multiple interpretations of the complex phenomena, of which one of the more popular ideas is based on *complex networks*. According to this perspective, behind each complex system there exists an underlying complex network which governs its behavior. Since the system is much more than the mere agglomeration of the individual parts, it makes the knowledge of the interactions between the individual components of paramount importance. Complex networks have been identified operating behind systems spanning a myriad of fields, including biology[1], organizational behavior, power distribution and political networks [2].

Networks have been an active area of research since the later half of the twentieth century, and not just in computer science. Sociologists studied behavior of crowds based on popular surveys and despite the small sample size, they observed effects like homophily [3] and six degrees of separation. Unlike sociologists, mathematicians and computer scientists took a more theoretical approach with a focus on studying and modeling networks[4], with not much importance placed on their real world applicability. This continued until the turn of the last century, when it all changed for the good with the inception of the internet [5] and the availability of other large scale data. This brought forward a whole new dimension to the world of complex networks — with new theories popping up explaining the large scale structure of networks.

We now shift our focus to the context of social networks where we focus on people and popular interactions. Man is a social animal, and right from the early days of human history, connections played a pivotal role in the society. Man exploited his links to his advantage for



(a) Undirected graph (b) Weighted undirected graph (c) Directed graph

Figure 1.1: Sample graphs each having 4 nodes and 4 edges

spreading ideas and beliefs giving rise to religions and popular beliefs. Fast forward a couple of centuries, the basic picture has largely remained the same. With the internet boom and the subsequent advent of online social networks, man has never been as connected as he is today. So, to study human behavior and to understand how information flows through a set of people, it's vital to know the underlying interaction network. Graph theory provides a formal but flexible framework to describe and manipulate networks, thereby enabling us to observe and formalize phenomena occurring in networks.

This work uses the language of graph theory to model information diffusion in networks to build a recommendation system which takes into account the topological features of the underlying network, thereby increasing its effectiveness.

1.1 Definitions

To understand a complex system, we first need to know how its components interact with each other. A graph is a catalog of a system's components often called *nodes* or *vertices* and the direct interactions between them, called *links* or *edges*.

The links in some cases can be directed; an edge (u, v) signifies that u is linked to v but not the other way around. The links can also be *weighted*, with weights representing the strength of interactions.

Mathematically, the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$represents a tuple of two sets — $\mathcal{V}$ the set of nodes and $\mathcal{E}$ the set of edges, where $\mathcal{E} = \{(u, v)|u, v \in \mathcal{V}\}$. A graph's $order$ and $size$ represent the number of nodes and edges respectively. In this work, the terms *network* and *graph* are used synonymously.

The neighbors of a node in a graph are the set of nodes that can be reached by visiting the outgoing edges of that node. Mathematically, the set of neighbors of a node $u \in \mathcal{V}$ in a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, is represented as $\Gamma(u) = \{v|v \in \mathcal{V}, (u, v) \in \mathcal{E}\}$. For example, in 1.1a, $\Gamma(2) = \{1, 3, 4\}$ while in 1.1c, $\Gamma(2) = \{4\}$

1.2 Graph Representations

Edges and neighborhood of nodes can be represented in different ways, out of which the following two are the most popular.

1.2.1 Adjacency Matrix

The adjacency matrix $\mathcal{A}(\mathcal{G})$ of a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$is a square matrix of the order $|\mathcal{V}|$ where the entries are defined as

$$\mathcal{A}(\mathcal{G})_{ij} = \begin{cases} 1, \ if \ (i,j) \in \mathcal{E} \\ 0, \ otherwise \end{cases}$$

For weighted networks $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{W})$, where each edge $(u, v) \in \mathcal{E}$ has a corresponding weight $w_{uv} \in \mathcal{W}$, the entries of the adjacency matrix reflects the weights.

$$\mathcal{A}(\mathcal{G})_{ij} = \begin{cases} w_{ij}, \ if \ (i,j) \in \mathcal{E} \\ 0, \ otherwise \end{cases}$$

It provides $\mathcal{O}(1)$ lookup for edges, but as a downside requires $\mathcal{O}(|\mathcal{V}|^2)$ space which becomes problematic for large graphs especially for the sparse ones.

The adjacency matrix of 1.1a is

$$\mathcal{A}(\mathcal{G}) = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

for 1.1b is

$$\mathcal{A}(\mathcal{G}) = \begin{bmatrix} 0 & 2 & 1 & 0 \\ 2 & 0 & 5 & 3 \\ 1 & 5 & 0 & 0 \\ 0 & 3 & 0 & 0 \end{bmatrix}$$

and for 1.1c

$$\mathcal{A}(\mathcal{G}) = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

1.2.2 Adjacency List

The adjacency list of a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$is a list of lists. There is exactly $|\mathcal{V}|$ lists, corresponding to each node where their neighboring nodes are listed.

Adjacency lists are more efficient than the adjacency matrices, taking $\mathcal{O}(\mathcal{V}+\mathcal{E})$ in place of $\mathcal{O}(|\mathcal{V}|^2)$ while the cost of edge lookups increase to $\mathcal{O}(degree)$ from $\mathcal{O}(1)$

For 1.1a, the adjacency list $\mathcal{L}(\mathcal{G})$ is

$$\mathcal{L}(\mathcal{G}) = \begin{Bmatrix} 1: & \{2, & 3\} \\ 2: & \{1, & 3, & 4\} \\ 3: & \{1, & 2\} \\ 4: & \{2\} \end{Bmatrix}$$

and for 1.1c it is,

$$\mathcal{L}(\mathcal{G}) = \begin{Bmatrix} 1: & \{2, & 3\} \\ 2: & \{4\} \\ 3: & \{2\} \\ 4: & \{\} \end{Bmatrix}$$

1.3 Degree and Degree Distribution

1.3.1 Degree

The degree of a node is the count of its neighbors. For weighted graphs, the weighted degree of a node represents the sum of the weights of outgoing edges of that node. For directed graphs, each node has a in, out and $total$ degrees, representing the number of incoming, outgoing and total neighbors.

Mathematically, in a graph $\mathcal{G}(\mathcal{V},\mathcal{E})$, for a node $u \in \mathcal{V}$, the degree is represented as $k_u = |\Gamma(u)|$. In the case of an adjacency matrix $\mathcal{A}$, $k_i = \sum_j \mathcal{A}_{ij}$ while in the adjacency list $\mathcal{L}$, $k_i = |\mathcal{L}_i|$.

$$\sum_i k_i = \begin{cases} 2|\mathcal{E}| \text{ if } \mathcal{G} \text{ is undirected} \\ |\mathcal{E}| \text{ if } \mathcal{G} \text{ is directed} \\ 2\sum_{ij} w_{ij} \text{ if } \mathcal{G} \text{ is weighted} \end{cases}$$

In 1.1a,

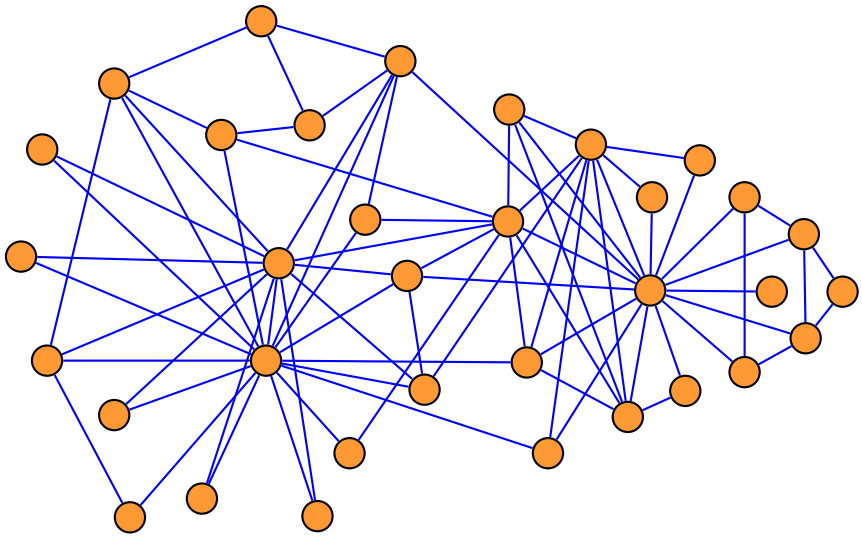
$$k_1 = 2, k_2 = 3, k_3 = 2, k_4 = 1 \text{ and } \sum_i k_i = 8 = 2|\mathcal{E}|$$

.

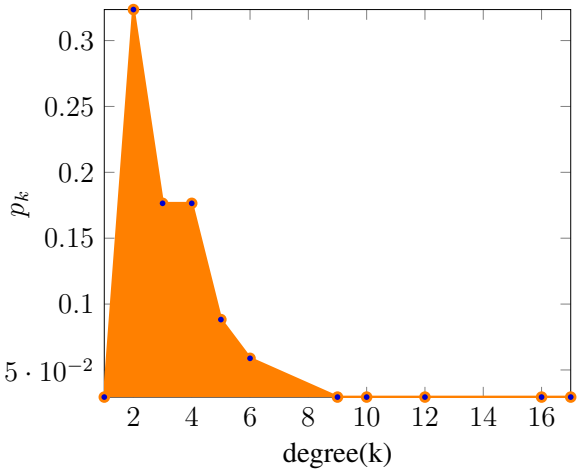
While in 1.1b,

$$k_1 = 3, k_2 = 10, k_3 = 6, k_4 = 3 \text{ and } \sum_i k_i = 22 = 2\sum_{ij} w_{ij}$$

.



(a) The network with 34 nodes and 78 edges



(b) Degree distribution of the network

Figure 1.2: Zachary's Karate Club

And in 1.1c,

$$k_1^{in} = 0, k_2^{in} = 2, k_3^{in} = 1, k_4^{in} = 1$$
$$k_1^{out} = 2, k_2^{out} = 1, k_3^{out} = 1, k_4^{out} = 0$$
$$k_1^{total} = 2, k_2^{total} = 3, k_3^{total} = 2, k_4^{total} = 1$$

1.3.2 Degree Distribution

The degree distribution, p_k , provides the probability that a randomly selected node in the network has degree k. Since p_k is a probability, it must be normalized, i.e $\sum_{k=1}^{\infty} p_k = 1$. Degree distributions are usually represented by histograms. For 1.2a, the degree distribution is: $p_1 = \frac{1}{4}, p_2 = \frac{2}{4}$ and $p_3 = \frac{1}{4}$ and $p_k = 0$ for $k > 3$.

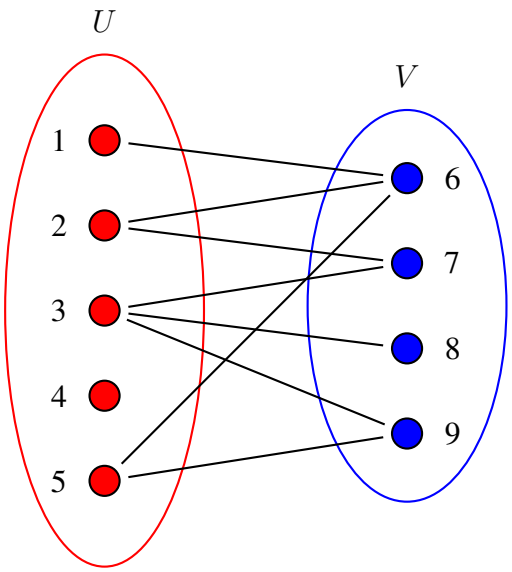


Figure 1.3: A small bipartite network with the disjoint node sets marked in red and blue respectively

The degree distribution of social networks follow a power law with a heavy tail[6]. Thus, $p_k \sim k^\gamma$ where γ lies between 2 and 3. So, there is a large number of nodes with small degrees and a few nodes called *hubs* with very high degrees.

1.4 Bipartite Graphs

When modeling relations between two different classes of objects, bipartite graphs very often arise naturally. For instance, a graph of football players and clubs, with an edge between a player and a club if the player has played for that club, is a natural example of an affiliation network. Other examples include predator prey networks and flower pollinator networks.

Bipartite graphs are a special class of graphs whose nodes can be divided into disjoint sets $\mathcal{V}_1$ and $\mathcal{V}_2$ such that each edge connects a node from $\mathcal{V}_1$ to a node in $\mathcal{V}_2$.

From a graph coloring perspective, the two sets of nodes in $\mathcal{V}_1$ and $\mathcal{V}_2$ can be thought of having two different colors, say red and blue respectively. The network shown in figure 1.3 is an example of a bipartite network.

1.5 PageRank

First introduced by Brin and Page in 1997[7], *PageRank* has become one of the most used centrality measures. Being based on a random surfer model, the *PageRank* score of a node u represents the probability that the random surfer ends up at u. The distribution of *PageRank* in networks follow a power law according to[8], i.e. a lot of nodes have a relatively low *PageRank* while a few nodes have very high values.

Mathematically, ***PageRank*** of a node u in a directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$is defined in a recursive manner as shown here.

$$PageRank(u) = \alpha \sum_{v \in \Gamma(u)} \frac{PageRank(v)}{k_v^{out}} + \beta$$

where α, β and k_v^{out} represent the damping factor, the initial bias and the out-degree of node v respectively. α and β are picked in a manner so that $\alpha + \beta = 1$.

PageRank can be computed in a variety of ways, including iteratively using the *power iteration* method, and using linear algebra. Here we focus on the former method, owing to the greater speed and memory efficiency over the latter method.

Let $PageRank^t(u)$ be the ***PageRank*** of node u of a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$at time t.

Initially at time $t = 0$,

$$PageRank^0(u) = \frac{1}{\mathcal{V}}, \qquad \forall u \in \mathcal{V}$$

.

And the ***PageRank*** scores at time $t + 1$ are updated as follows

$$PageRank^{t+1}(u) = \alpha \sum_{v \in \Gamma(u)} \frac{PageRank^t(v)}{k_v^{out}} + \beta, \qquad \forall u \in \mathcal{V}$$

The ***PageRank*** scores are normalized, that is the sum of the ***PageRank*** scores of all the nodes is 1.

1.6 Independent Cascade Model

Proposed by[9], the Independent Cascade model is one of the more popular models to model information diffusion. Given a network $\mathcal{G}(\mathcal{V}, \mathcal{E})$, each node classified as either *active* (if it has the information) or *inactive*; and each edge $(u, v) \in \mathcal{E}$ has a certain probability, called the *propagation probability* p_{uv}— the probability of the information passing from u to v — associated with it. Given a seed set of initial adopters $\mathcal{S}$, the Independent Cascade model models the spread as described below.

At every timestamp t, each newly active node u gets a single chance to pass on the information to each of its inactive neighbors $v(v \in \Gamma(v))$ with the probability p_{uv}, i.e. the propagation probability associated with the edge (u, v). If u succeeds, v becomes active at timestamp $t + 1$. However, if u fails, it cannot try again in the subsequent timestamps. This process continues until there are no further activations. The formal algorithm is described in **??**.

Algorithm 1: Independent-Cascade $(\mathcal{G}(\mathcal{V}, \mathcal{E}), \mathcal{S})$

```
begin
    spread_count ← 0
    foreach u ∈ V do
        u.color ← WHITE
    Q ← φ
    foreach seed ∈ S do
        Q ← Q ∪ {seed}
    spread_happens ← TRUE
    while spread_happens do
        u ← Q.dequeue()
        spread_count ← spread_count + 1
        foreach v ∈ Γ(u) do
            if v.color = WHITE then
                v.color ← BLACK
                r ← random(0, 1)
                if r ≤ p_uv then
                    spread_count ← spread_count + 1
                    spread_happens ← TRUE
    return spread_count / |V|
```

1.7 Community Structure in Networks

1.7.1 Introduction

Most networks of interest display *community structure*, i.e., their nodes are organized into groups, called *communities*, *clusters* or *modules*. In Fig. 1.4 we show a collaboration network of scientists working at the Santa Fe Institute (SFI) in Santa Fe, New Mexico where the nodes are scientists, and edges join coauthors. Edges are concentrated within groups of vertices representing scientists working on the same research topic, where collaborations are more natural. Likewise, communities could represent proteins with similar function in protein-protein interaction networks, groups of friends in social networks, websites on similar topics on the Web graph, and so on.

Identifying communities may offer insight on how the network is organized. It allows us to focus on regions having some degree of autonomy within the graph. It helps to classify the nodes, based on their role with respect to the communities they belong to. For instance we can distinguish nodes totally embedded within their clusters from vertices at the boundary of the clusters, which may act as *brokers* between the modules and, in that case, could play a major role both in holding the modules together and in the dynamics of spreading processes across the network.

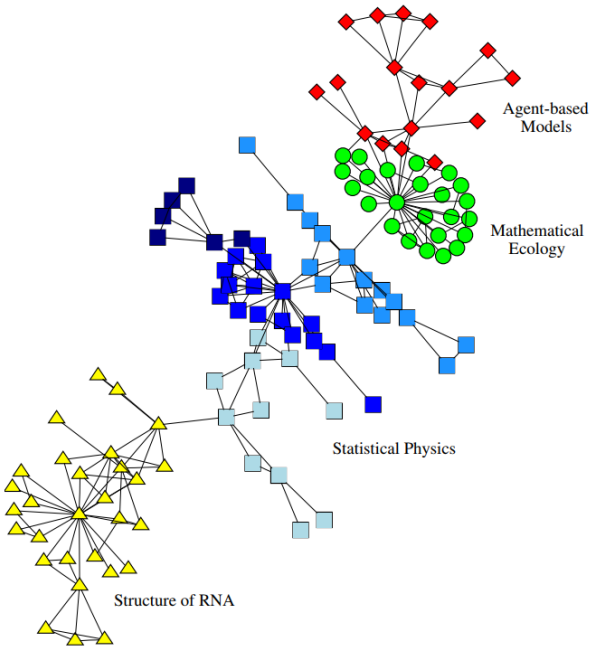
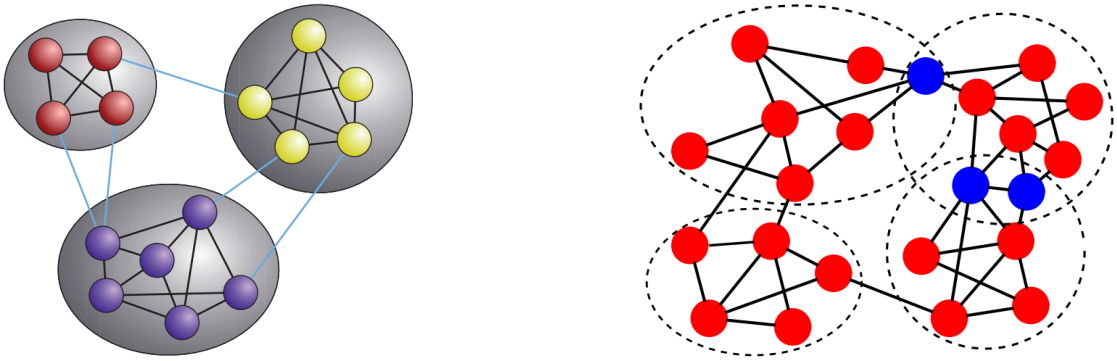


Figure 1.4: Santa Fe Institute collaboration network[10]



(a) A disjoint cover

(b) An overlapping cover

Figure 1.5: Examples of the two types of covers

1.7.2 Formal Definitions

1.7.2.1 Cover

For a given graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, a cover $\mathbb{C} = \{C_1, C_2, ..., C_k\}$ such that $\bigcup_i C_i = \mathcal{V}$, is a multi-set of sets of nodes where nodes in the same set lie in the same community. Depending on the membership behavior of nodes, covers can be classified into two types —disjoint (like in figure 1.5a) or overlapping (like in figure 1.5b). In the case of a *disjoint* cover, each node belongs to exactly one community, ie. $\forall i, j$, $C_i \cap C_j = \phi$. Whereas in a *overlapping* cover, a node may coexist in multiple communities, or $\exists\ i, j$ such that, $C_i \cap C_j \neq \phi$.

Community Detection algorithms depending on their nature, look to unveil either disjoint or overlapping covers in a network. We will now look at some popular community detection methods.

1.7.2.2 Modularity

In a randomly wired network the connection pattern between the nodes is expected to be uniform, independent of the network's degree distribution. Consequently these networks are not expected to display systematic local density fluctuations that we could interpret as *communities*. Using this idea, we can come up with the definition of *modularity*, that allows us to measure the quality of the cover. It measures the normalized difference of the fraction of edges that lie inside the communities and the same fraction had the graph been truly random with the same degree distribution.

Formally, in a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, and a disjoint cover $\mathbb{C}$, modularity ($\mathcal{Q}$) is defined as

$$\mathcal{Q} = \sum_{c \in \mathbb{C}} \left[\frac{m_c}{|\mathcal{E}|} - \left(\frac{\sum_{u \in c} k_u}{2|\mathcal{E}|} \right)^2 \right] \tag{1.1}$$

where m_c is the number of *intra-community* edges in community c, and k_u is the degree of node u.

By definition, $\mathcal{Q} \in [0, 1]$.

The higher is $\mathcal{Q}$ for a partition, the better is the corresponding community structure. We can use modularity to decide which of the many partitions predicted by a hierarchical method offers the best community structure, selecting the one for which M is maximal. It can also be shown that modularity maximization is also a NP-Hard problem.

1.7.3 Community Detection Methods

1.7.3.1 Graph Bisection

This is one of the simplest methods of finding groups in graphs. We divide the graph into two non-overlapping subgraphs, such that the number of links between the nodes in the two groups is minimized. The widely used Kerninghan-Lin algorithm uses the approach.

We can solve the graph bisection problem by inspecting all possible divisions into two groups and choosing the one with the smallest cut size. To determine the computational cost of this brute force approach, we note that a network with n nodes into groups of n_1 and n_2 nodes is

$$\frac{n!}{n_1|!\, n_2!} \tag{1.2}$$

Using Stirling's formula $n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$, we can write (1.2) as

$$\frac{n!}{n_1|!\, n_2!} \approx \frac{\sqrt{2\pi n} \left(\frac{n}{e}\right)^n}{\sqrt{2\pi n_1} \left(\frac{n_1}{e}\right)^{n_1} \sqrt{2\pi n_2} \left(\frac{n_2}{e}\right)^{n_2}} \approx \frac{n^{n+\frac{1}{2}}}{n_1^{n_1+\frac{1}{2}}\, n_2^{n_2+\frac{1}{2}}} \tag{1.3}$$

To simplify the problem, let's set the goal of dividing the network in two equal sized portions, then $n_1 = n_2 = \frac{n}{2}$, in this case 1.3 becomes

$$\frac{2^{n+1}}{\sqrt{n}} = e^{(n+1)\,ln2 - \frac{1}{2}lnN} \tag{1.4}$$

indicating that the number of bisections increases exponentially with the size of the network.

To illustrate the implications of (1.4) consider a network with ten nodes which we bisect into two subgraphs of size $n_1 = n_2 = 5$. According to (1.2) we need to check 252 bisections to find the one with the smallest cut size.

Let us assume that our computer can inspect these 252 bisections in one millisecond (10^{-3} sec). If we next wish to bisect a network with a hundred nodes into groups with $n_1 = n_2 = 50$, according to (1.2) we need to check approximately 10^{29} divisions, requiring about 10^{16} years on the same computer. Therefore our brute-force strategy is bound to fail, being impossible to inspect all bisections for even a modest size network.

When the number of groups is unknown, the problem of finding communities becomes even harder. The computational complexity becomes worse than exponential in the size of the graph. This leads us to no choice but to settle with sub-optimal results obtained from various heuristics.

1.7.3.2 Modularity Based Methods

Given the difficulty of conventional methods to uncover community structure, we look to harness the notion that covers with a higher modularity score are better than the ones with lower scores. Or, in other words, for a given network the cover with maximum modularity corresponds to the *optimal* community structure.

The maximum modularity hypothesis is the starting point of several community detection algorithms, each seeking the partition with the largest modularity. In principle we could identify the best partition by checking M for all possible partitions, selecting the one for which M is largest. Given, however, the exceptionally large number of partitions, this brute- force approach is computationally not feasible. Next we discuss an algorithm that finds partitions with close to maximal M, while bypassing the need to inspect all partitions.

Newman's Greedy Algorithm - The first modularity maximization algorithm, proposed by Newman[11], iteratively joins pairs of communities if the move increases the partition's modularity.

Algorithm 2: Newman-Greedy($\mathcal{G}(\mathcal{V}, \mathcal{E})$)

```
begin
    Assign each node to a community of its own
    For each pair of connected communities, compute ΔQ if we merge them
    Identify the community pair for which ΔQ is the largest and merge them
    Repeat steps 3 and 4 until all nodes merge into a single community
    Return the cover for which Q is maximum
```

Since the calculation of $\Delta\mathcal{Q}$ can be done in constant time, the steps 3 and 4 requires $O(m)$ computations. After deciding which communities to merge, the update of the matrix can be done in a worst- case time $O(n)$. Since the algorithm requires $n-1$ community mergers, its complexity is $O((m+n)n)$. For sparse networks, the computation cost comes down to $O(n^2)$. Optimized implementations reduce the algorithm's complexity to $O(nlog^2n)$.

Vanilla Louvain - The $O(n^2)$ computational complexity of the greedy algorithm can be prohibitive for very large networks. A modularity optimization algorithm with better scalability was proposed by Blondel and collaborators[12].

The Louvain algorithm consists of two phases that are repeated iteratively. The phases are formally described in algorithms 3 and 4.

The key step in this method is the efficient computation of $\Delta\mathcal{Q}$. It is calculated as shown

$$\Delta\mathcal{Q} = \left[\frac{\Sigma_{in}+2k_{i,in}}{2\mathcal{W}} - \left(\frac{\Sigma_{tot}+2k_i}{2\mathcal{W}}\right)^2\right] - \left[\frac{\Sigma_{in}}{2\mathcal{W}} - \left(\frac{\Sigma_{tot}}{2\mathcal{W}}\right)^2 - \left(\frac{k_i}{2\mathcal{W}}\right)^2\right] \tag{1.5}$$

where Σ_{in} is the sum of the weights of the links inside community C; Σ_{tot} is the sum of the link weights of all nodes in C; k_i is the sum of the weights of the links incident to node i; $k_{i,in}$ is the sum of the weights of the links from i to nodes in C and $\mathcal{W}$ is the sum of the weights of all links in the network.

The Louvain method is more limited by storage demands than by computational time. The number of computations scale linearly with m, most time consuming first pass. With subsequent passes over a decreasing number of nodes and links, the complexity of the algorithm is at most $O(m)$. It therefore allows us to identify communities in networks with millions of nodes. It is widely considered to be the current state-of-the-art.

Algorithm 3: Louvain Method - Phase I - Modularity Optimization

```
1 begin
2     foreach u ∈ 𝒱 do
3         comm[u] ← u
4     𝒬 ← 0
5     repeat
6         foreach community c_u ∈ 𝒢 do
7             best_neighbor_comm ← argmax_{c_v ∈ Γ'(c_u)} {Δ𝒬 when c_u joins c_v using (1.5) }
8             𝒬 ← 𝒬 + max_{c_v ∈ Γ'(c_u)} {Δ𝒬 when c_u joins c_v}
9     until no improvement in Q
```

Bipartite Louvain - The Louvain algorithm works in the same manner for any network, meaning that it doesn't take into account the topology or features associated with the network. While this approach makes the method generic and flexible, it does come with a couple of caveats.

Algorithm 4: Louvain Method - Phase II - Community Aggregation

```
1  begin
2      V' ← φ
3      E' ← φ
4      foreach community c in G do
5          V' ← V' ∪ {c}
6      foreach community c_1 in G do
7          foreach community c_2 in G do
8              if (c_1, c_2) ∈ E then
9                  E' ← E' ∪ {(c_1, c_2)}
10                 w'(c_1, c_2) ← sum of edge weights between c_1 and c_2 in G
11     V ← V'
12     E ← E'
```

In the heart of the method, lies the computation of a quality metric, which is usually Newman's *modularity*. The effectiveness of the method is sensitive to this choice of quality function. While Newman's modularity is almost universally accepted as a fair and unbiased quality measure for covers, but in some cases, like in our work, there is a need of something that is more specialized.

In this work, we are dealing with bipartite networks like the one in figure 1.3, where all the edges span between the two disjoint set of nodes. The definition of Newman's modularity doesn't take into account this feature while creating the null model. This leads to a problem, and as a consequence to this, the quality of the cover takes a hit. Hence, we look to utilize a specialized definition of *modularity*, presented by *Barber*, which redefines modularity ($\mathcal{Q}_b$) in a bipartite network $\mathcal{G}(\mathcal{V}, \mathcal{E})$where $\mathcal{V} = V_R \cup V_B$ and $V_R \cap V_B = \phi$.

$$\mathcal{Q} = \sum_{c \in \mathbb{C}} \left[\frac{m_c}{|\mathcal{E}|} - \frac{r_c \, b_c}{|\mathcal{E}|^2} \right] \tag{1.6}$$

where m_c is the number of intra-community edges in community c; r_c and b_c is the sum of degrees of red and blue nodes in the cluster.

Since the definition of the modularity is different, we cannot use the old expression for $\Delta\mathcal{Q}$ as defined in (1.5). We modify the expression as

$$\Delta\mathcal{Q}_b = \left(\frac{d(c_1, c_2)}{|\mathcal{E}|} - \frac{d_r(c_1)d_b(c_2) + d_r(c_2)d_b(c_1)}{|\mathcal{E}|^2} \right) \tag{1.7}$$

where $d(c_1, c_2)$ is the number of edges spanning clusters c_1 and c_2; d_r and d_c is the sum of degrees of red and blue nodes in the cluster.

To find communities using this bipartite variant of modularity, we change the algorithm 3 to use the (1.7) instead of (1.5) in line 7.

Illustration As an example, consider the bipartite graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$where $V_R = \{R_0, R_1, R_2, R_3\}$ and $V_B = \{B_0, B_1, B_2, B_3\}$ as shown in the figure 1.6. The self loops indicate the degrees of the

Algorithm 5: Louvain Method - Modified Phase I - Modularity Optimization

begin
  foreach $u \in \mathcal{V}$ **do**
    $comm[u] \leftarrow u$
  $\mathcal{Q}_b \leftarrow 0$
  repeat
    foreach *community* $c_u \in \mathcal{G}$ **do**
      $best\_neighbor\_comm \leftarrow \underset{c_v \in \Gamma'(c_u)}{argmax}\{\Delta\mathcal{Q}_b \text{ when } c_u \text{ joins } c_v \text{ using (1.7) }\}$
      $\mathcal{Q} \leftarrow \mathcal{Q}_b + \underset{c_v \in \Gamma'(c_u)}{max}\{\Delta\mathcal{Q}_b \text{ when } c_u \text{ joins } c_v\}$
  until *no improvement in* Q

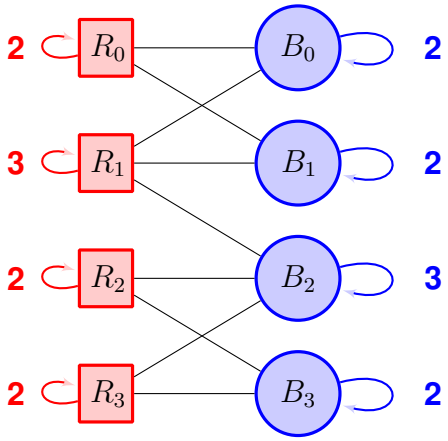


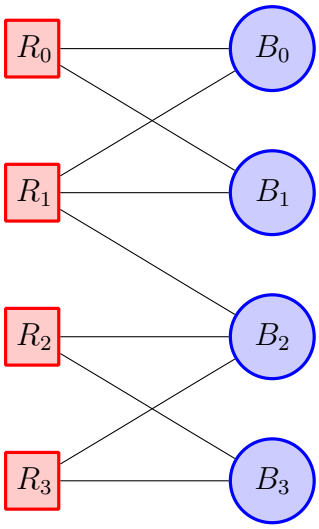
Figure 1.6: Bipartite graph example

nodes. See figure 1.7 for a walk-through.

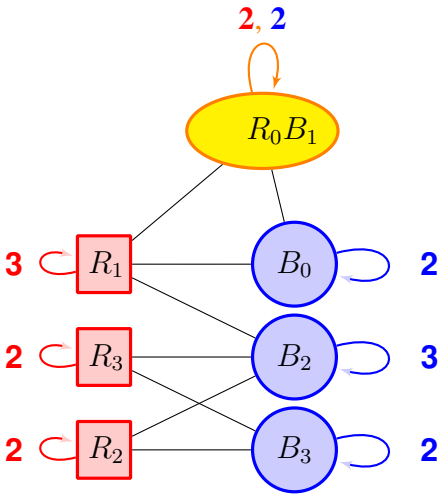
1.8 Recommender Systems

1.8.1 Introduction

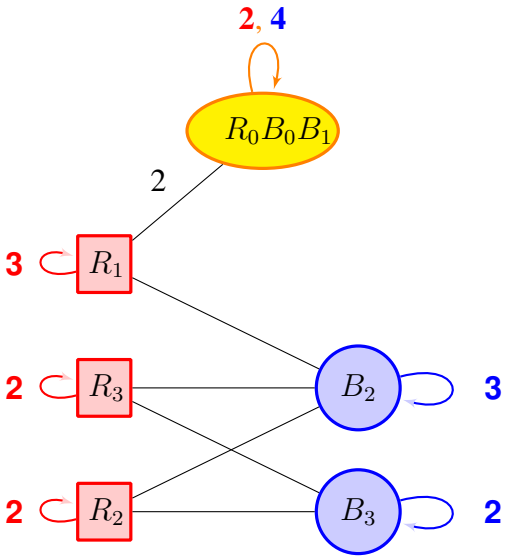
With the advent of the Internet, if a user wants a certain service or product, a myriad of browsing options is available but with the arrival of choices, comes confusion and hence the dire need of recommender systems. We see recommender systems in action, everywhere around us Starting from e-commerce sites like Amazon, recommending us products based on our search and purchase histories to the social networking sites like Facebook recommending us friends based on our social interactions. Recommender systems guide the users to their required or desired products by making guesses from the data available like browsing data, the items on which the user has shown interest before or sometimes by simply asking the users for their requirements. Thus on the basis of how the recommender system makes the guess, it is broadly divided into few different types.



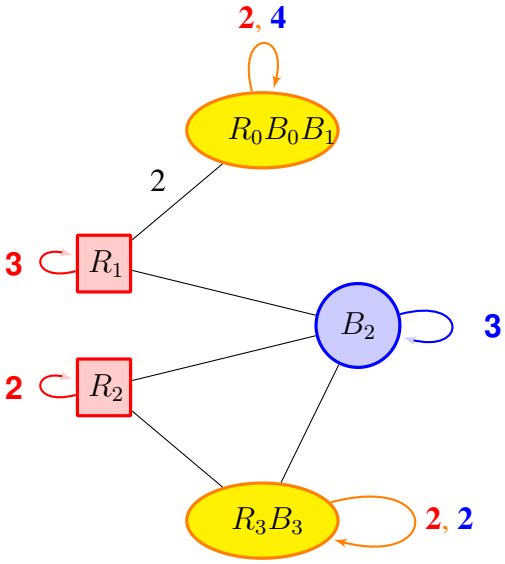
(a) Initial graph, $\Omega_b = 0$



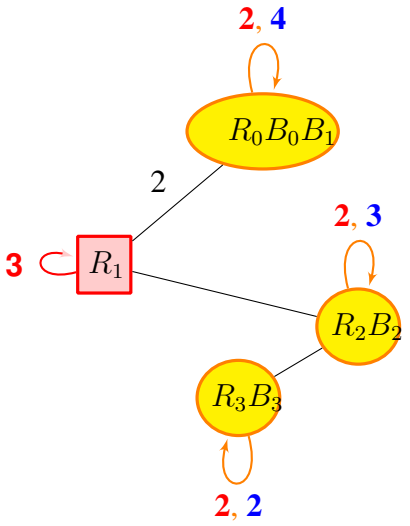
(b) R_0 and B_1 merges, $\Omega_b = 0.0617$



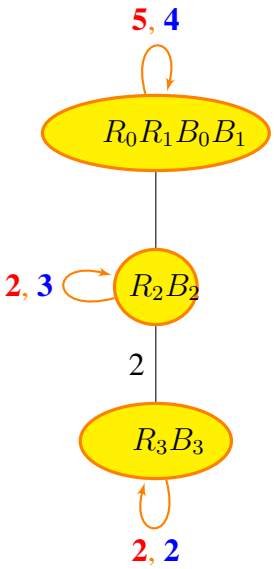
(c) R_0B_1 and B_0 merges, $\Omega_b = 0.1235$



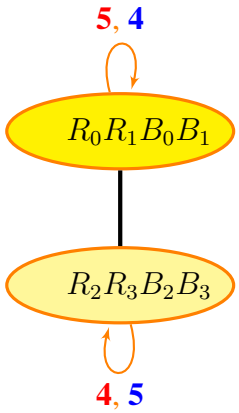
(d) R_3 and B_3 merges, $\Omega_b = 0.1852$



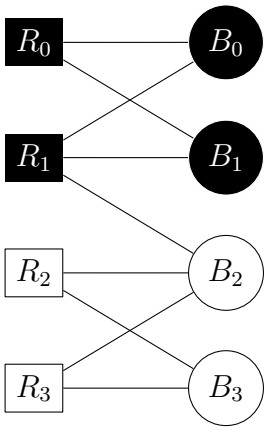
(e) R_2 and B_2 merges, $\Omega_b = 0.2469$



(f) R_1 and $R_0B_0B_1$ merges, $\Omega_b = 0.321$



(g) R_2B_2 and R_3B_3 merges, $\mathcal{Q}_b = 0.4198$



(h) Final network, $\mathcal{Q}_b = 0.4198$

Figure 1.7: Bipartite Louvain in action

In *collaborative filtering* the idea is to find similar users to the target user, and recommend him/her the items which has been liked by similar users. *Content based filtering* ensures features of the items liked by the target user is mapped to the features of items that are available. The system then recommends the similar featured items. In *Knowledge based filtering* the recommendations are based on explicitly specified user requirements. Instead of using historical rating or buying data, external knowledge bases and constraints are used to create the recommendation.

In our project we use collaborative filtering as the main basis and try to extend the idea to suit our needs and to improve the effectiveness of the recommender system.

1.8.2 Collaborative Filtering

1.8.2.1 Introduction

Collaborative filtering models use the collaborative power of the ratings provided by multiple users to make recommendations. The governing principle behind this is that the unspecified ratings can be imputed because the observed ratings are often highly correlated across various users and items.

For example, consider two users named $Alice$ and Bob, who have very similar tastes. If the ratings, which both have specified, are very similar, then their similarity can be identified and then be used to make inferences about incomplete values of one user with help of mentioned value of other user. But in the real-world sites, often, the problem becomes much more non-trivial due to the huge number of items and users, and a few instances of ratings or reviews given by the users to the items. The figure 1.8 shows the typical rating behavior of users. The distribution represents a *power law*, where there are a few popular items which a large fraction of users rate, but for most of the items, the number of ratings is small.

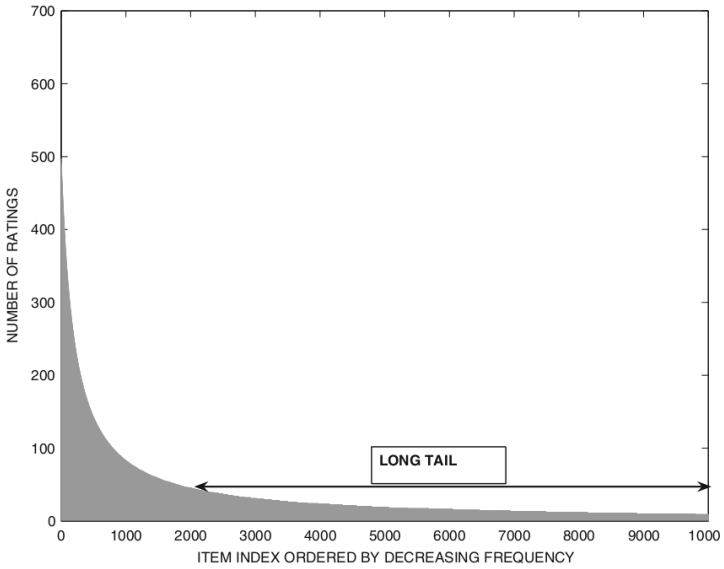


Figure 1.8: Distribution of typical rating frequencies

1.8.2.2 Types Of Collaborative Filtering

Memory-based methods - also referred to as *neighborhood based* collaborative filtering algorithms. These were among the earliest collaborative filtering algorithms, in which the ratings of user-item combinations are predicted on the basis of their neighborhoods.

These neighborhoods can be defined in one of two ways - in *user-based collaborative filtering*, the ratings provided by like-minded users of a target user *Alice* are used in order to make the recommendations for*Alice*. Thus, the basic idea is to determine users, who are similar to *Alice*, and recommend ratings for the unobserved ratings of *Alice* by computing weighted averages of the ratings of this peer group. In general, the k most similar users to A can be used to make rating predictions. Similarity functions are computed between the rows of the ratings matrix to discover similar users.

Item-based collaborative filtering - determines a set S of items that are most similar to target item X. The ratings in item set S, which are specified by *Alice*, are used to predict whether the user *Alice* will like item X. Similarity functions are computed between the columns of the ratings matrix to discover similar items.

The advantages of memory-based techniques are that they are simple to implement and the resulting recommendations are often easy to explain. But on the other hand, memory-based algorithms do not work very well with sparse ratings matrices. In other words, such methods might lack full coverage of rating predictions. Nevertheless, the lack of coverage is often not an issue, when only the top-k items are required.

Model-based methods: In model-based methods, machine learning and data mining methods are used in the context of predictive models. In cases where the model is parameterized, the parameters of this model are learned within the context of an optimization framework. Some examples of such model-based methods include decision trees, rule-based models, Bayesian

Item / User	I_1	I_2	I_3	I_4	I_5	I_6	Mean Rating
u_1	7	6	7	4	5	4	5.5
u_2	6	7	**?**	4	3	4	4.8
u_3	**?**	3	3	1	1	**?**	2
u_4	1	2	2	3	3	4	2.5
u_5	1	**?**	1	2	3	3	2

Table 1.1: Utility matrix of users

methods and latent factor models. Many of these methods, such as latent factor models, have a high level of coverage even for sparse ratings matrices.

1.8.2.3 User Based Collaborative System

User-based neighborhoods are defined in order to identify similar users to the target user for whom the rating predictions are being computed. In order to determine the neighborhood of the target user u, her similarity to all the other users is computed. Therefore, a similarity function needs to be defined between the ratings specified by users. Such a similarity computation is tricky because different users may have different scales of ratings. One user might be biased toward liking most items, whereas another user not liking most of the items. Furthermore, different users may have rated different items. Therefore, many similarity measures exist to choose from according to the need of individual algorithms.

We have a utility matrix at our disposal with U as the set of all users and I as the set of all items, showcasing the ratings given by $u \in U$ to the items in I.

1.8.2.4 Similarity Measures

For the $m \times n$ ratings matrix $R = [r_{uj}]$ with m users and n items, let I_u denote the set of item indices for which ratings have been specified by user (row) u. For example, for user u_3, $I_3 = \{2, 3, 4, 5\}$. Therefore, the set of items rated by both users u and v is given by $I_u \cap Iv$. For example, for users u_2 and u_3, $I_2 = \{1, 2, 4, 5, 6\}$ and $I_3 = \{2, 3, 4, 5\}$and $I_u \cap I_v = \{2, 4, 5\}$. The set $I_u \cap I_v$ defines the mutually observed ratings, which are used to compute the similarity between the users u and v for neighborhood computation. There are several measures that compute similarity and have different significances.

One measure that captures the similarity between the rating vectors of two users is the *Pearson correlation coefficient*. Because$I_u \cap I_v$represents the set of item indices for which both user u and user v have specified ratings, the coefficient is computed only on this set of items. The first step is to compute the mean rating $\mu_u \; \forall u \in U$. Now strictly speaking, *Pearson correlation* requires the μ_u to be calculated over only the common items shared with the user v, $I_u \cap I_v$. But sometimes, a general mean over all the items rated by u, I_u is used to calculate the mean.

Hence, two equations of finding mean are popular.

$$\mu_u = \frac{\sum_{k \in I_u} r_{uk}}{|I_u|}, \qquad \forall u \in U \tag{1.8}$$

$$\mu_u = \frac{\sum_{k \in I_u \cap I_v} r_{uk}}{|I_u \cap I_v|}, \quad \forall u \in U \tag{1.9}$$

Traditionally the definition of Pearson correlation coefficient mandates the use of Equation (1.9). But Equation (1.8) is quite common as calculation of mean is done only once. Also if two users have only one mutual rating then using Equation (1.8) helps to give a non-zero result. It cannot be argued that one method is better than the other, hence we use the Equation (1.8) throughout our algorithm.

Next using the mean ratings we calculate the *Pearson Correlation coefficient*, which is given as follows:

$$Pearson(u,v) = \frac{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u).(r_{vk} - \mu_v)}{\sqrt{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u)^2} \sqrt{\sum_{k \in I_u \cap I_v} (r_{vk} - \mu_v)^2}} \tag{1.10}$$

Other variants of the similarity function are also used in practice. One variant is to use the *cosine similarity* on the raw ratings rather than the mean centered ratings, though in general the Pearson correlation is preferable to the raw cosine because of the bias adjustment effect of mean-centering. This adjustment accounts for the fact that different users exhibit different levels of generosity in their global rating patterns. The *Cosine similarity* is given as follows:

$$RawCosine(u,v) = \frac{\sum_{k \in I_u \cap I_v} r_{uk}.r_{vk}}{\sqrt{\sum_{k \in I_u \cap I_v} r_{uk}^2} \sqrt{\sum_{k \in I_u \cap I_v} r_{vk}^2}} \tag{1.11}$$

The reliability of the similarity function Sim(u, v) is often affected by the number of common ratings $|Iu \cap Iv|$ between users u and v. When the two users have only a small number of ratings in common, the similarity function should be reduced with a discount factor to de-emphasize the importance of that user pair. This method is referred to as *significance weighting*. The discount factor is used, when the number of common rating between two user is less than a particular threshold β. The value of the discount factor is given by $\frac{min|I_u \cap I_v|, \beta}{\beta}$, and it always lies in the range $[0, 1]$. Therefore, the *Discounted similarity,* $DiscountedSim(u,v)$ is given by the following:

$$DiscountedSim(u,v) = Sim(u,v).\frac{min|I_u \cap I_v|, \beta}{\beta} \tag{1.12}$$

1.8.2.5 Illustration

Here is a small example to show the calculation of similarity functions. If we want to find the similarity score between the user 1 and 3 then we first look at the means $\mu_1$and μ_3, and the ratings of both user on common items:

User	u_1		u_2		u_3		u_4		u_5	
	RC	P	RC	P	RC	P	RC	P	RC	P
u_1	1	1	0.97	0.70	0.95	0.89	0.83	-0.89	0.80	-0.82
u_2	0.97	0.70	1	1	0.98	0.93	0.81	-0.71	0.83	-0.89
u_3	0.95	0.89	0.98	0.93	1	1	0.78	-1	0.64	-0.81
u_4	0.83	-0.89	0.81	-0.71	0.78	-1	1	1	0.98	0.87
u_5	0.80	-0.82	0.83	-0.89	0.64	-0.81	0.98	0.87	1	1

Table 1.2: User-user similarity matrix. RC and P represents the $RawCosine$ and $Pearson$ correlation coefficient similarities respectively.

$$Pearson(1,3) = \frac{(6-5.5).(3-2)+(7-5.5).(3-2)+(4-5.5).(1-2)+(5-5.5).(1-2)}{\sqrt{1.5^2+1.5^2+(-1.5)^2+(-0.5)^2}\sqrt{1^2+1^2+(-1)^2+(-1)^2}}$$
$$= 0.894$$

For cosine the raw rating of users are used:

$$RawCosine(1,3) = \frac{6.3+7.3+4.1+5.1}{\sqrt{6^2+7^2+4^2+5^2}\sqrt{3^2+3^2+1^2+1^2}}$$
$$= 0.956$$

Table 1.2 has the pairwise similarities for the $RawCosine$ and $Pearson$ and similarities of the users in utility matrix in Table 1.1.

1.8.2.6 Rating Prediction

Different users may provide ratings on different scales. One user might rate all items highly, whereas another user might rate all items negatively. The raw ratings, therefore, need to be *mean-centered* in row-wise fashion, before determining the (weighted) average rating of the peer group. The mean-centered rating s_{uj} of user u for item j is given by:

$$s_{uj} = r_{uj} - \mu_u, \quad \forall u \in U \tag{1.13}$$

The predicted rating of user u on item j, $\hat{r_{uj}}$is given by adding the mean rating of user u to the mean centered predicted rating by finding the weighted average of the ratings given by similar users, where the weights are the similarity score. The the final predicted score is given by:

$$\hat{r_{uj}} = \mu_u + \frac{\sum_{v \in P_u(j)}(r_{vj} - \mu_v)\, Sim(u,v)}{\sum_{v \in P_u(j)} |Sim(u,v)|} \tag{1.14}$$

where $P_u(j)$ is the set of $k - closest$ user to the target user u, who have specified ratings for item j.

For user u_3, the peer group may consists of user u_1 and u_2 because the rest of the users have negative similarity score, or are dissimilar to user u_2. Thus using the values in Table 1.2, we can make prediction for item I_1 for user u_3 as follows:

$$\begin{aligned}\hat{r}_{31} &= 2 + \frac{1.5\,.\,0.894 + 1.2\,.\,0.939}{0.894 + 0.939} \\ &\approx 3.35\end{aligned}$$

considering $P_u(j) = \{u_1, u_2\}$.

The similarity measures come into play only when there is intersection of reviews/ratings given by the users. Say user *Alice* has rated for items 1 and 2, and user *Bob* has rated the items 4 and 6 then we have no way to find the similarity between these users. Extending this idea we can never recommend an item to *Alice* which has only been rated by users like *Bob* with whom there is zero intersection. The high sparsity of utility matrix makes this a very common occurrence.

Pearson correlation coefficient as we discussed before, works better than Cosine similarity due to the consideration of mean centered ratings. But even so, there are time times when Pearson Correlation fails. If one of the vectors of user item rating consists of only one rating, the Pearson correlation will come out as zero. Also if we are using (1.9), with one mutually rated item, the similarity is undefined. At times like these Cosine similarity could have given more useful results. The discounted similarity measure accounts for a useful information which should be taken into consideration, but due to the sparse nature of ratings finding intersection itself becomes a rarity.

In the coming chapter we will discuss our proposed method in detail.

Chapter 2

Network Aware Recommender System

2.1 Clustering in Collaborative Filtering

The main problem with neighborhood-based methods in collaborative filtering is the complexity of the offline phase, which can be quite significant when the number of users or the number of items is very large. For example, when the number of users m is of the order of a few hundred million, the $O(m^2n)$ running time of a user-based method will become impractical even for occasional offline computation. Consider the case where $m = 10^8$ and $n = 100$. In such a case, $O(m^2n) = O(10^{18})$ operations will be required. If we make the conservative assumption that each operation requires an elementary machine cycle, a 10 GHz computer will require 10^8 seconds, which is approximately 115.74 days. Clearly, such an approach will not be very practical from a scalability point of view.

The application of clustering techniques also reduces the sparsity . Different clustering strategies can be performed based on users and items. In general, clustering users (or items) results in creating sub-matrices of the entire user-item rating matrix. Then classical CF algorithms (user-based and item-base d) can be used to generate recommendation based on these sub-matrices. When user-based CF is applied on user clusters, the neighbors of the active user are users in the same user cluster. If item-based CF is used, the ranking of an item is based on the items which are rated by users in the same user cluster.

Most methods clusters the rows for user-based collaborative filtering. In item-based methods, it would be necessary to cluster the columns. The approach is exactly similar except that it is applied to the columns rather than the rows. Some of these methods are user-based methods, whereas others are item-based methods. A number of co-clustering methods can be used to cluster rows and columns simultaneously.

2.2 Zomato

Zomato is a restaurant search and discovery service founded in 2008 by Deepinder Goyal and Pankaj Chaddah. It currently operates in 23 countries, including India, Australia and the United States. It provides information and reviews on restaurants, including images of menus where the restaurant does not have its own website. It has invested in many related restaurant search and recommendation networks like Gastronaucci and Urbanspoon and currently is competent with Yelp, Zagat and OpenTable.

It works towards a simple motto of ensuring, nobody should have a bad meal. It helps users to discover great places around their locality, which matches their taste, budget and other preferences. This not only helps users, but also restaurants by providing them the much needed market and thus reducing their management costs. Other than recommending restaurants to the user, Zomato also provides services like table-booking, food ordering and mobile based app. The idea that Zomato is working globally, gives us a great opportunity to look at the food habit trends across the world.

Zomato ensures that users can make good and non-biased food choices by making thousands of public reviews of dining experience available publicly. Each restaurant has its own page where Zomato shows all the varied information regarding a restaurant like cost for two, area, cousins, opening hours, contact details of the restaurants and other facilities like valet parking, vegetarian services, sports activities, pools, delivery services, bars etc provided by the restaurants. The users share their experience of dining, service, ambiance, worth of money and photos of the place and food being served. Zomato regularly verifies specific users to help others to identify genuine reviews time to time.

2.3 Crawling Zomato

Limitations posed by Zomato on crawling the Zomato API, calls for the need of crawling data from the site pages itself. Also the API does not provide any information regarding the underlying user-restaurant networks networks, so, we had to design our own scraping tool which gathers the information we need. The scraper is written exclusively in Python with the BeautifulSoup and Selenium libraries. We start off with the top 10 popular restaurants of Kolkata which include Arsalan, Barbecue Nation, Chili's Grill and Bar among others. We go on to discover the user-restaurant network in other states of India – Bangalore, Pune, Hyderabad, Delhi, Chennai – and also in other cities like Boston, New York and Chicago. We scrape the restaurant pages and user pages from the site and form the network by connecting the users to the restaurant when a user has rated a restaurant and connect all other restaurants reviewed by the user and so on.

We start with a list of restaurant and visit individual restaurant. We store the information like name of restaurant, number of reviews, rating, cost for two, geological location and cuisines served. Then we systematically visit all the reviews given to the restaurant, and using load more

we scrape all the reviews in the subsequent pages. We store the reviewer's name, time of review rating given and the review text.

2.3.1 Underlying Bipartite Network

On the popular restaurant search engine Zomato, the users and the restaurants form a bipartite network where the two disjoint sets are the sets of users and restaurants. A directed link is established between an user u and and a restaurant r if u writes a review of r. Each review has the following components — the user details including the name, user_id and a link to the profile of the user and the review details which has the time stamp, the rating (from 0 to 5 in increments of 0.5) and the review text.

2.4 Our Work

Following the collaborative filtering concept, we try to inculcate the idea of similarity to find out new choices of restaurants for users. As we saw similarity can be traced between both, users and restaurants. Initially, we have a bipartite network of restaurants and users which has connecting edges if a user has reviewed a certain restaurant. Using the methods of clustering we form bipartite clusters consisting of users and restaurants. A cluster in a network signifies a dense region and it is certain that the number of intra cluster edges will be higher than the number of inter-cluster edges. From the model that we achieve after running our clustering algorithms, we can conclude that the types of similarity between users can be mapped into 3 broad categories - users who are two hop away and in same cluster, users who are more than two hop away and in same cluster and users who are in the other cluster.

For the second categories, we tweak our similarity finding measures. In basic collaborative filtering method, the similarity is drawn only between users who have an intersection of restaurant or are two hop away form each other. But if a restaurant is more than 3 hop away from the target user, ie. it does not have any user who has rated the target restaurant and has not mutually rated a common restaurant, then we can never recommend it to the user, which is a popular occurrence as the utility matrix is extremely sparse.

In 2.2a, here user U can't be recommended the restaurant R. Thus we can extract the above tree like structure of users and restaurants where each hop is represented by the consecutive levels consisting of either restaurants or users. From the figure 2.2a we can see that the users two hop away from each other have at least one restaurant in common.

For users more than 2 hop away, and in the same cluster, we use a measure like cumulative Pearson correlation to find the similarity of users which we explain above.

If the restaurant lies in a different cluster we don't find an immediate predictive rating to keep the working of our algorithm efficient.

Say our target user is U and the target restaurant is R. The set of users who are more than two hop away and rated for the target item is $U\_far = U_{31}, U_{32}$. For all the users which have mutually rated restaurant, a Pearson Correlation similarity can be found. We consider the absolute value of Pearson which lies between 0 and 1. Considering the mutual restaurants we find the Pearson Correlation similarity and form the network shown in figure 2.2a. Using these as similarity between two users we try to extend the similarity transitively to the users with whom we don't have the direct Pearson Score. We weight the edges using the reciprocal of Pearson score and find the shortest path between the target user and all the users in set U_far. Finding the reciprocal ensures that the shortest path gives us the path which has highest similarity. Also the Pearson score of zero is not taken into account while forming the network and thus on taking reciprocal we won't get undefined value. Using Dijkstra algorithm we find the shortest path, having the highest similarity. Taking U as target node and U_31 as destination node, $U\_path\_k = 1, 0.98$ comes out to be the Pearson scores in the shortest path. In this path we look at the Pearson Correlation and find a weighted average of Pearson score using a decay measure (λ) as weights. We assign decay to the corresponding edge as

$$decay = \lambda^{hops\ from\ target\ user} \tag{2.1}$$

$$CumulativePearson_k = \frac{\Sigma_i(decay_i * Pearson_i)}{\Sigma_i(decay_i)}, i \in U\_path, k \in U\_far \tag{2.2}$$

Now we take these $CumulativePearson_i$ as the similarity between the user U and the users $i \in U\_far$ and find the weighted average of the ratings given by these users.

$$FinalScore = \frac{\Sigma_i(CumulativePearson_i * Rating_i)}{\Sigma_i(CumulativePearson_i)}, i \in U\_far \tag{2.3}$$

2.4.1 Illustration

Taking user U and restaurant R which is 5 hops away, and users in $U_far = \{U_{31}, U_{32}, U_{33}\}$ we find the shortest paths as mentioned above, and we consider the Pearson Correlation values from figure 2.2b to calculate the final score, with $\lambda = \frac{1}{2}$: Thus for U and U_{31},

$$CumulativePearson = \frac{1 * (1) + 0.98 * 0.5}{1 + 0.5} \tag{2.4}$$

$$= 0.99 \tag{2.5}$$

And for U and U_{32},

$$CumulativePearson = \frac{1 * 1 + 0.89 * 0.5}{1 + 0.5} \tag{2.6}$$

$$= 0.94 \tag{2.7}$$

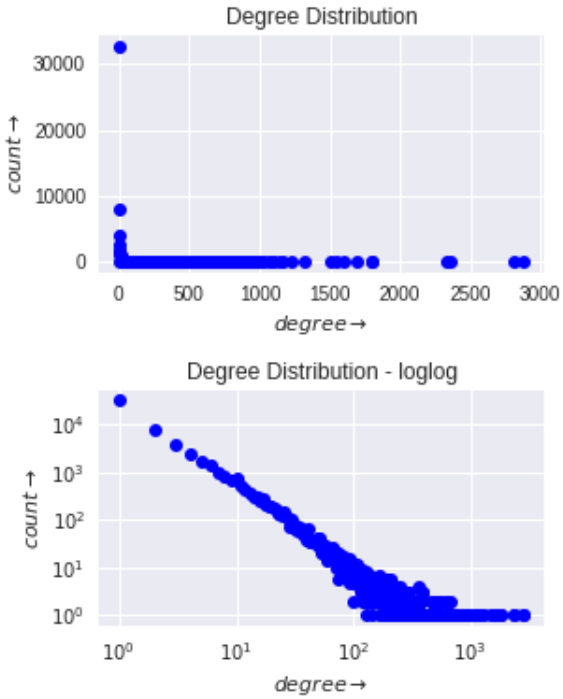


Figure 2.1: Degree distribution of the network

Thus final predicted rating of user U to restaurant R is

$$= \frac{0.99 * 5 + 0.94 * 4}{0.99 + 0.94} \tag{2.8}$$

$$= 4.51 \tag{2.9}$$

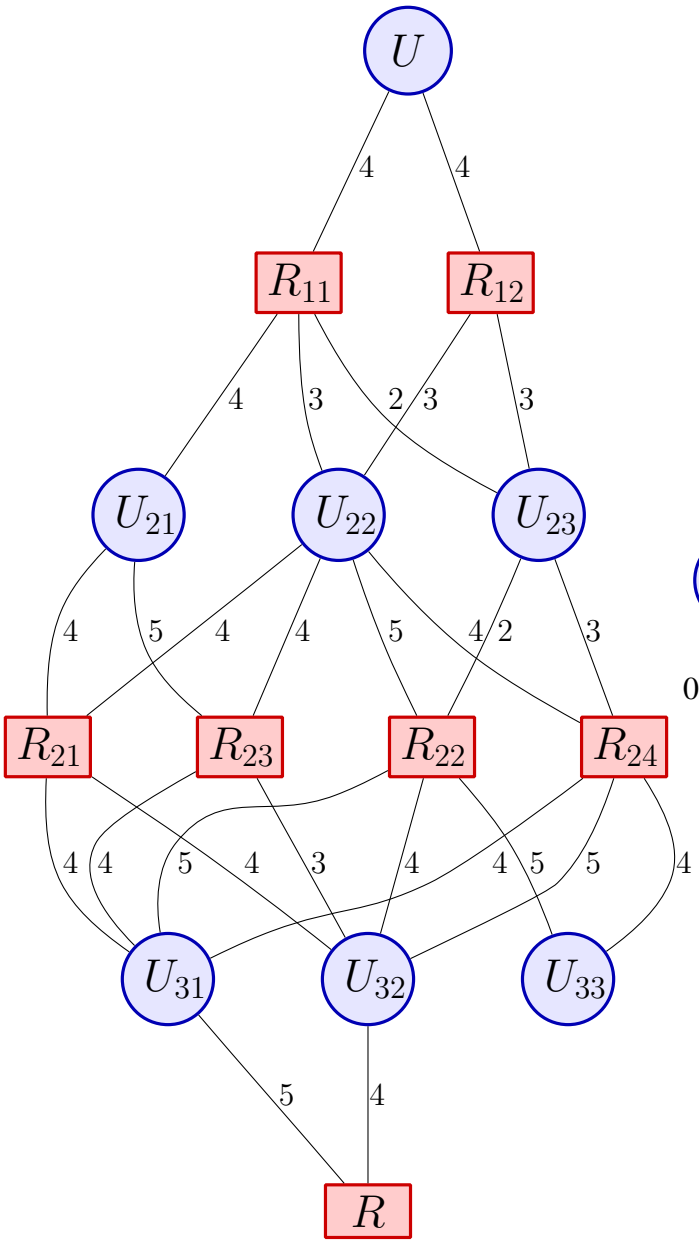
$$\approx 4.5 \tag{2.10}$$

Property	Value
Order ($\|V\|$)	$60,483$
Size ($\|E\|$)	$255,185$
#Restaurants	$3,351$
#Users	$57,132$
#Components	99

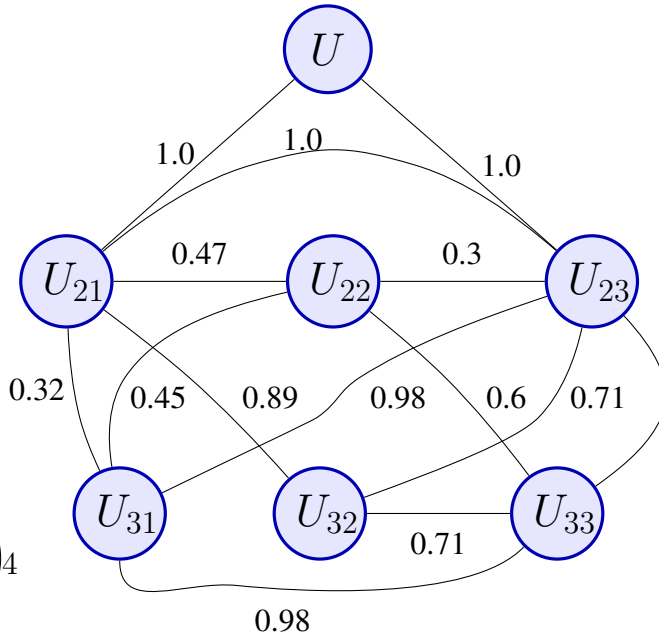
Table 2.1: Network statistics

2.5 Results

We use this approach to assign ratings to the restaurants in the test data. Comparing with normal Collaborative Filtering, we obtain the following results in 2.3.

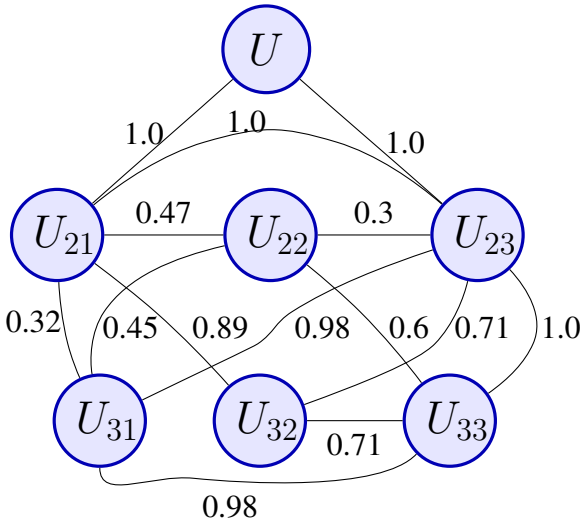


(a) Tree Structure

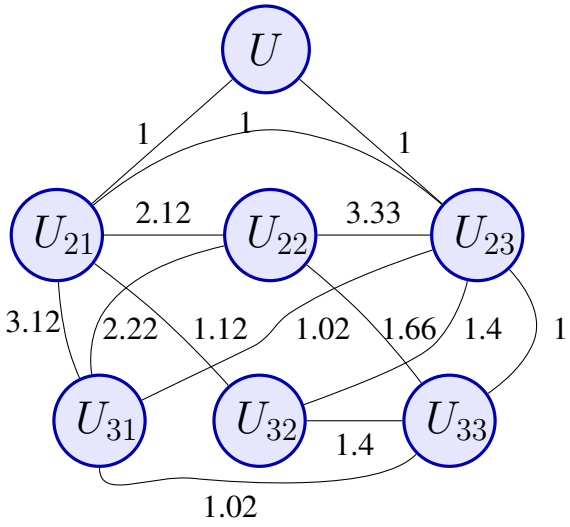


(b) Similarity graph

Figure 2.2: A worked out example

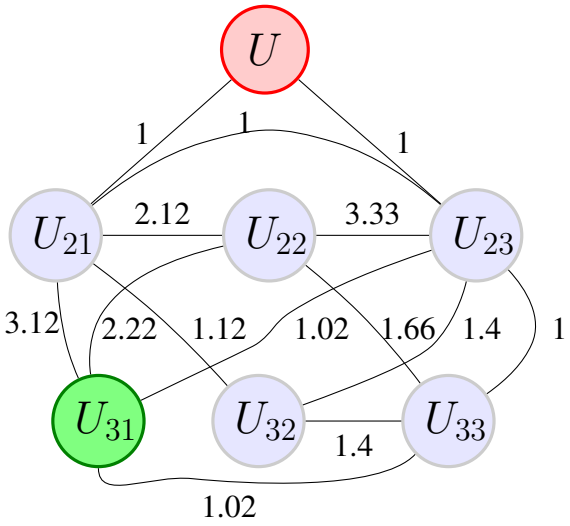


(a) Pearson similarity graph



(b) Inverse similarity graph

Figure 2.3: Similarity and inverse similarity graphs



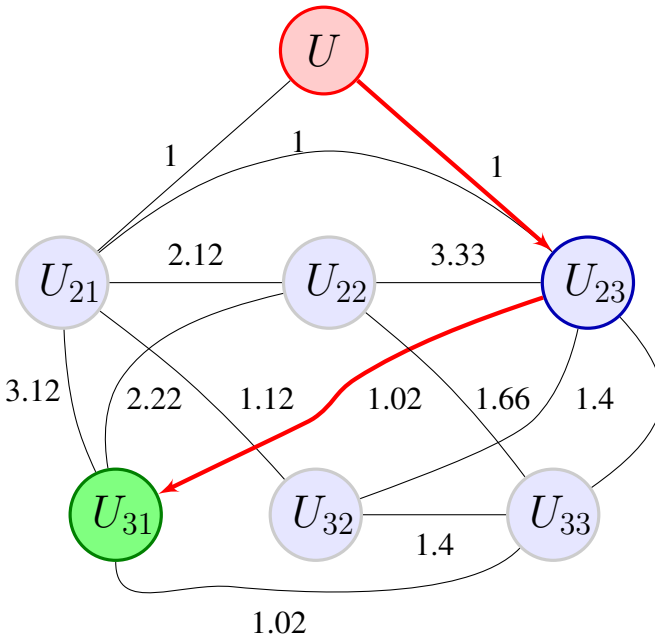


Figure 2.4: Shortest path from U to U_{31} highlighted in red

Year	#Reviews
2016	103, 538
2015	76, 071
2014	34, 953
2017	23, 423
2013	13, 705
2012	3, 412
2011	366
2010	32
2009	6

Month	#Reviews
October	26, 571
July	23, 078
September	22, 336
June	22, 045
January	21, 726
August	21, 115
May	21, 089
December	20, 978
March	20, 475
February	19, 398
November	18, 997
April	17, 698

Table 2.2: Review statistics

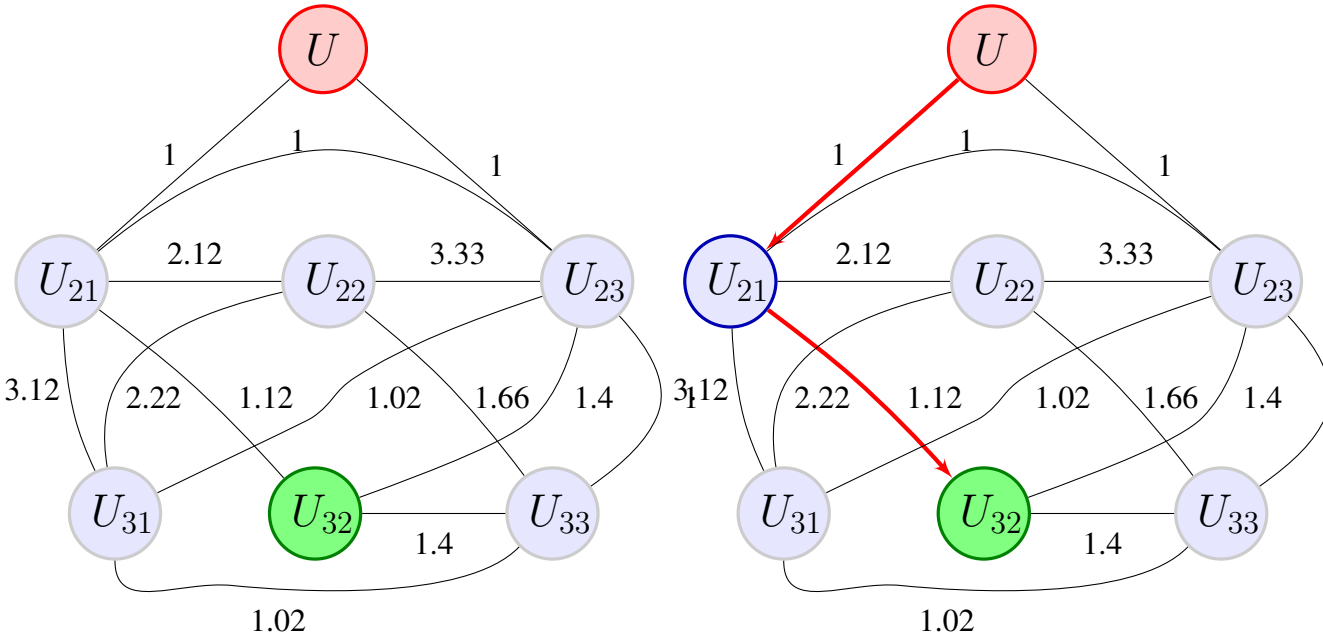


Figure 2.5: Shortest path from U to U_{32} highlighted in red

2.6 Conclusion and Future Work

Thus empirically we have shown that our approach of combining Collaborative Filtering with bipartite clustering beats the vanilla Collaborative Filtering approach.

This work has a lot of scope for extensions, out of which we are specifying a handful. Currently, we use a disjoint community detection technique but overlapping clustering will give us more meaningful clusters. Markov Chains and other random walk techniques can be employed to find similar users.

City	Collaborative Filtering	Our Approach
	MSE	MSE
Kolkata	9.2	8.3
Hyderabad	8.5	8.4
Pune	12.1	10
New Delhi	11.1	12.5
Chennai	17.9	15.4
Bangalore	15.21	16
Mumbai	9	11
New York	10.5	8.9
Boston	15.21	14
Chicago	20	18

Table 2.3: Results

Bibliography

[1] James Coleman, Elihu Katz, and Herbert Menzel. The diffusion of an innovation among physicians. *Sociometry*, 20(4):253–270, 1957.

[2] Lada A Adamic and Natalie Glance. The political blogosphere and the 2004 us election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, pages 36–43. ACM, 2005.

[3] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, pages 415–444, 2001.

[4] Paul Erdős and Alfréd Rényi. On the strength of connectedness of a random graph. *Acta Mathematica Hungarica*, 12(1-2):261–267, 1961.

[5] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[6] Mark EJ Newman. Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351, 2005.

[7] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[8] Luca Becchetti and Carlos Castillo. The distribution of pagerank follows a power-law only for particular values of the damping factor. In *Proceedings of the 15th international conference on World Wide Web*, pages 941–942. ACM, 2006.

[9] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146. ACM, 2003.

[10] Mark EJ Newman. The structure of scientific collaboration networks. *Proceedings of the National Academy of Sciences*, 98(2):404–409, 2001.

[11] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[12] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.