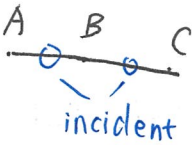
Week 2 图算法与联通性

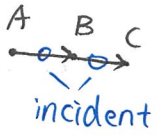
1. Connectivity in Graphs

1.1 Concept Review

① Adjacent：节点间有边 

② incident：边共享节点.（有向则需方向相同）





③ Walk：一系列依次访问的边（节点、边可重复）

· Open walk：起终不同 

· Close walk：起终相同 

④ ~~Cycle~~ Path：节点和边都不重复的 walk

⑤ Cycle：起终点相同的 path

⑥ Random Walk：每一步随机选邻居

$$W_{i,x} = \frac{A_{ix}}{d_i}$$ 权重 A_{ix}，d_i 为 $\sum_x A_{ix}$

$\sum_x W_{i,x} = 1, \forall i,j\ w_{ij} \geq 0$

nodes similarity & node importance 从 i 访问 x 的概率

Page Rank 基础

1.2 Connectivity

① connected graph：任意两点间有 path

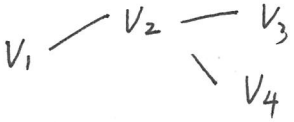
② Disconnected graph：存在点间无 path

③ 有向图 { weak connectivity：忽略边方向后，任意两点存在 path
strong connectivity：任意两点存在有向路径（双向可达）

1.3 Components { 在无向图中：connected subgraph
directed path: { strongly connected：$\forall (u,v) \in V$，存在 path $u \to v$ $v \to u$
weekly connected：把边方向取消是 connected subgraph

1.4 n-hop neighbourhood of a node 是 node set：within n hops distance from node

dist ≤ n.



V_2 是 V_1 1 hop $\{V_2, V_3, V_4\}$ 是 V_1 的 2 hop

1.5 Diameter：图中最大的 shortest path

$\max_{(v_i,v_j) \in V \times V} l_{ij}$

l_{ij} 是 shortest path from i to j

2. Special Graphs

2.1 Tree: 无环无向图. 任意两点恰有一条path

怎么证 $|E| \geq |V|-1$ 不然连不起来 $|E| \leq |V|-2$. 再加一条成环

$|V| = |E|+1$

Forest: A set of disconnected trees

2.2 Spanning Trees: 对一个connected graph, spanning tree 是包含其所有node的 tree 无环无向图 $|V| = |E|+1$

对一个 weighted graph, 自然有 minimum spanning tree. 拥有最小权重

MST 成本最低连接方案. 网络布线. 修路. 桥

2.3 Complete Graph 任意两点都有边 $E = \frac{|V| \times (|V|-1)}{2}$

2.4 Bipartite Graph 二分图. 节点可分为两组. 边只在组边、组内无边

2.5 Regular Graphs 所有节点 degree 相同 理想化

2.6 Bridges: 边 whose removal will increase the number of connected components

帮找 community Detection: (u,v). 先去掉, 从u开始, 若u还能到v则非bridge

3. Graph Traversal Algorithms (Assumption: strongly connected)

3.1 DFS: ① 访问 v_i, 选择一个邻居 v_j, 递归进行DFS. 直到没有未访问邻居. 回溯

Stack 后进先出 标记 v_i 访问过 (未访问

BFS:

Queue 先进先出 ① 访问 v_i, v_j 全入列. 前面走一个, 入列其邻居

爬虫: BFS √ large network, only a portion, a representative budget vs bias in details

3.2 Djikstra (非负权重图) 因为对访问过的节点, 若有负边使path↓也不会再进行更新/重新处理

1. 初始化 distance[start] = 0 distance[others] = Infinity. Set = {unvisited}
2. While unvisited set != empty set:
   a. current node = node in unvisited with smallest distance
   b. currentnode = endnode: break. 找到了
   c. for neibors of current node. distance[neibor] = min(distance[neibor], Weight(current, neibour) + distance[current
   d. remove current from unvisited.

3.3 Prim: MST

1. 随机选一个节点加入
2. 找最小权重边. (树内节点和树外节点)
3. 拉该边和另一个节点下水
4. 重复直到所有节点加入

undirected

数组: 线性扫描 $O(V^2)$ $E \approx V^2$

堆优化实现 = 双堆 $O((V+E)\log V)$ $E \ll V^2$

4. Network Measures

4.1 Centrality Measures 正比于在网络中的重要度

- geometric
- spectral
- path-based

4.1.1 Geometric Measure

① Degree Centrality (连接越多越重要)

$C_d(v_i) = d_i$ $C_d^{in}(v_i) = d_i^{in}$ $C_d^{out} = d_i^{out}$

$C_d(v_i) = d_i^{in} + d_i^{out}$

Indegree 更重要被关注难

不同平台用户数不同, 归一化

$$C_d^{norm}(v_i) = \frac{d_i}{n-1}$$ ← 最多 $n-1$

$$C_d^{max}(v_i) = \frac{d_i}{\max_j d_j}$$ ← 最大的

$$C_d^{sum}(v_i) = \frac{d_i}{\sum_j d_j} = \frac{d_i}{2|E|}$$ 边条数两倍

~~$C_d^{sum}(v_i) = \frac{d_i}{\sum}$~~

4.1.2 Closeness Centrality (重要节点能更快速到达其他节点)

$$C_c(v_i) = \frac{1}{\frac{1}{n-1}\sum_{v_j \in V} l_{ij}}$$

$\sum_{v_j \in V} l_{ij}$ ← node i 到其他所有节点的 shortest path 和, 归一化.

n 个, 但 $l_{ii} = 0$.

无法应用于非 strongly connected graph: disconnect 节点到其他节点距离为无穷.

when $l_{ij} \to \infty$, $C_c(v_i) \to \cancel{\infty}\ 0$

⇓ 于是使用

4.1.3 Harmonic Centrality (任意图, 含 un strongly connected graph)

$$C_{har}(v_i) = \sum_{v_j \in V} \frac{1}{l_{ij}}$$

$\frac{1}{\infty} = 0$ 不连通也能算

Imple. ~~$\forall v_i$~~ for $v_i \in V$:

Dijkstra (v_i, other nodes)

compute $C_h(v_i) = \sum 1/l_{ij}$

Rank $C_h(v_i)$

4.2 Spectral Measure

为何引入？ Degree Centrality 可以买粉

希望考虑 朋友/关注者的中心性

拥有重要朋友比拥有更多朋友更能说明重要性

4.2.1 Eigenvector Centrality 节点重要性取决于其邻居的重要性

$$Ce(v_i) = \frac{1}{\lambda}\sum_{j=1}^{n} A_{ji}\, Ce(v_j)$$

A_{ji}：从 j 到 i 的 connectivity/weight

怎么算？ Power Iteration

1. Intialize $Ce(v_1) \dots Ce(v_n)$ to 1
2. Update based on equation $Ce(v_i) \leftarrow \sum_{j=1}^{n} A_{ji}\, Ce(v_j)$
3. Normalization $Ce(v_i) \leftarrow \frac{Ce(v_i)}{\sqrt{\sum_{j=1}^{n} (Ce(v_j))^2}}$

   避免数值爆炸/消失
4. 重复 2, 3 直到 don't change much

向量：$Ce = 1$

$$Ce = \frac{1}{\lambda} A^T Ce \iff A^T Ce = \lambda Ce$$

无向则 $A^T = A$

Ce 是邻接矩阵 A^T 特征向量

$O(n^3)$

存在问题：有向图中中心性可能为 0 只看 A_{ji} 出入向，忽略 α 向

当节点在弱连接图分量，只有入边，没有出边时，对其他节点，其 centrality 传不出去，给其他节点逐渐调零，直到都成 0

4.2.2 Katz Centrality

$$C_{katz}(v_i) = \alpha \sum_{j=1}^{n} A_{ji}\, C_{katz}(v_j) + \beta$$

$\alpha = 0$ 全看 β，α 大，β 影响小 控制 term

β：给所有节点基础 centrality

$$C_{katz} = \alpha A^T C_{katz} + \beta 1$$

$$C_{katz} = \beta (I - \alpha A^T)^{-1} 1$$

bias

Power Iteration

1. Initialization：所有节点 centrality 设为 1
2. Update $C_{katz}(v_i) \leftarrow \sum_{j=1}^{n} A_{ji}\, C_{katz}(v_j) + \frac{\beta}{\alpha}$
3. Normalize $C_{katz}(v_i) \leftarrow \frac{C_{katz}(v_i)}{\sum (C_{katz}(v_j))^2}$

   向量的 L-2 norm

问题：① 权威节点问题：高 centrality 将所有 centrality 传给所有出边邻居，不合理，被知人认识不一定出名，且没有按 d^{out} 归一化

其实 eigen 也有这样问题

4.2.3 Page Rank 将传递的 centrality 按 out degree 分配，每个邻居只占一部分

$$Cp(v_i) = \alpha \sum_{j=1}^{n} \frac{A_{ji}\, Cp(v_j)}{d_j^{out}} + \beta \quad \text{其中 } \beta = 1-\alpha$$

可以理解成访问 j 的 prob

Random walk 里从 j 到 i 的 prob

这里要注意，如果是 weighted，~~原来~~要变成 $\frac{W_{ji}}{\sum_k W_{jk}}$

如果只有出边，没有入边，random walk会被卡住，所以引入随机walk的概率 α 访问neighbour ~~$\alpha(1-\alpha)$~~

$1-\alpha$ 的p访问任意一点

prob of visiting node v_i

$$p_i = \text{从邻居来+乱来} = \alpha \sum_{v_j \in N(v_i)} p_j \frac{A_{ji}}{d_j^{out}} + (1-\alpha) \sum_{j \in |V|}^{n} p_j \frac{1}{n} = \alpha \sum_{j=1}^{N} A_{ji} \frac{p_j}{d_j^{out}} + (1-\alpha)$$

↑ 从n个点来

Power Iteration

1. Init: $C_p(v_1) \cdots C_p(v_n) = \frac{1}{n}$ 相同prob
2. Update $C_p(v_i) \leftarrow$ ~~$C_p(v_i)$~~ $\alpha \sum_{j=1}^{n} A_{ji} \frac{C_p(v_j)}{d_j^{out}} + \beta$
3. Norm $C_p(v_i) \leftarrow \frac{C_p(v_i)}{\sum_j C_p(v_j)}$ 不是 l_2-norm
4. 直到收敛

5. Transitivity and Reciprocity

5.1 Transitivity 朋友的朋友也是朋友 ⇒denser graph → complete graph

理解并连接密度，评估community 三角形.

Global Clustering Coefficient

$$C = \frac{\#\text{paths of length2 that have the third edge}}{\#\text{paths of length 2}}$$

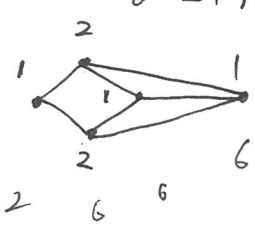
对一个undirected graph，一个三角形有6条 paths of length 2

$$= \frac{\text{number of triangles} \times 3}{\text{number of triplet of nodes}}$$

A→B→C, A→C→B, ... 一共6条path

an ordered set of 3 nodes

上下约去2即可



一条三角形3个triplet

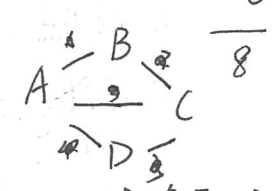
2个三角形 $\frac{2\times3}{2\times3+?}$ ← 剔去三角形的triplet

一个triplet

v_i 和 v_j 连. v_j 和 v_k 连.

$(v_i, v_j, v_k) = (v_k, v_j, v_i)$ 但这有两条length2 paths

即 v_k — v_j — v_i 形



$$C = \frac{2\text{个三角形}\times3}{6 \text{ 2个三角形} + (A,B,D) (B,C,D)} = \frac{2\times3}{2\times3+2}$$

123 132 231 213 312 321 6

Local Clustering Coefficient 一般用在 undirected graphs

$$C(v_i) = \frac{\#\text{pairs of neighbors of } v_i \text{ that are connected}}{\#\text{ possible pairs of neighbors of } v_i}$$

Average clustering coefficient

$$C(G) = \frac{1}{|V|}\sum_{v_i \in V} C(v_i)$$

个 $\frac{d_i(d_i-1)}{2}$

5.2 Reciprocity 互惠性 简化的传递性，考虑长度为2的闭环

$$R = \frac{2\times \#\text{Reciprocal node pairs}}{\#\text{ of edges}}$$ → $a \longleftrightarrow b$

6. Social Balance

6.1 signed graph (-1/+1) 正边多于负边 大多是+++/+--

6.2 Social Balance Theroem 对任意环，若边值乘积为正，该环是社会平衡的

三角形(i,j,k)平衡充要 $w_{ij}\cdot w_{jk}\cdot w_{ki} \geq 0$.

7. Similarity

7.1 Vertex Similarity 两个节点共享的邻居 共享邻居越多越相似

① Jaccard Similarity = $\frac{|N(v_i)\cap N(v_j)|}{|N(v_i)\cup N(v_j)|}$ ← 邻居交集size / 邻居并集size

② Cosine Similarity = $\frac{|N(v_i)\cap N(v_j)|}{\sqrt{|N(v_i)||N(v_j)|}}$ 几何平均.

7.2 Link Prediction based on vertex similarity

1. 对节点v，计算v与所有未连接节点相似度
2. 选相似度 Tok-k
3. 推送

无法处理新用户（推受欢迎、自己选、个人资料推荐）

Network Models

还需看lab和代码

1. Motivation

1.1 Challenge: 巨大且复杂 ⇒ 直接分析非常困难且成本高

1.2 目的: ① 理解机制 ② 简化分析 ③ 提供解释 ④ 可控实验 ⑤ 预测增长

Step 1: 分析并理解现实世界网络属性

2: 提出数学模型, 生成能捕捉这种属性的网络

2. Properties of real-world networks

① Degree distribution $P(k) = \frac{N_k}{N} = \frac{\#\text{ degree } k \text{ nodes}}{\#\text{ nodes}}$

大多数用户 own 少量 links. 少数拥有大量 (长尾效应)

Power-law distribution $f(k) = a \cdot k^{-b}$ [2, 3]

检验方法: 在 $\ln f(k)$ ↑ → $\ln(k)$ 作图, 呈直线则符合 power law

$f(k)$ 为 degree $= k$ 的 nodes 数量(比例). b 是 power-law exponent

power law intercept

$\ln f(k) = \cancel{-b\ln} \ln a + (-b)\ln(k)$ 一次函数

具有 power-law 分布的网络 scale-free : same functional form at all scales

→ 没有典型均值(因为有超级枢纽在)

取度小于 k 部分仍符合 power law

当被测量的 metric 可被视为某种"流行度", power law seems to dominate

马太效应

少数节点拥有绝大多数 link. 这种分布使得网络放大还是缩小, 结构规律都一样

② Clustering coefficient

现实中: high clustering coefficient : friendships are highly transistive

小网络(圈子)倾向大 coefficient

③ Average Path Length : The world is small

现实中. $\forall$ 2 members . connected via short paths

Six Degree of Separation Stanley Milgram 1960

网络↑ degree of separation ↓

④ Number of connected component is small : 存在 a large giant component

link 增长快于 size

有大牛把小群体连起来

3. Evolution of Network Models

1. Random Graphs

Assumption: 节点间的边是随机生成的

一个点可能有 $n-1$ nodes

expected degree $(n-1)p$

expected edges $C_n^2 p$

$G(n, p)$: 节点数 n, ~~边数~~ . 每条边以 prob p 出现 gnp_random_graph(n, p, seed, directed)

$G(n, m)$: n nodes m edges

$|\Omega| = \binom{\frac{n}{2}}{m}$ 总计 C_n^2 可能边, 选 m 个

↑ 从中选一个

n 很大时, 二者表现相似

$m = \binom{2}{n} p$

2. Giant Component 一个巨大的connected component. 在random graphs中 $\begin{cases} p=0 \to \text{size of GC is 0} \\ p=1 \to \text{size of GC is } n \end{cases}$

Phase Transition 相变 当 average degree $c=1$ 或 $p=\frac{1}{n-1}$ 发生.

$\begin{cases} c<1: \text{孤立小簇} & \text{short path lengths} \\ c=1: \text{GC出现, diameter 达到顶峰} & \text{path lengths long} \\ c>1: \text{几乎所有节点相连, 直径缩小} & \text{path lengths} \downarrow \end{cases}$

对Random Graphs $P(d_v=d)=\binom{n-1}{d}p^d(1-p)^{n-1-d}$ Binomial Distribution

从除自己以外 $n-1$ 个选 d 个连通 $\downarrow n\to\infty$

Possion degree distribution 不是 power law

$C(v)=\frac{\#\text{ v-neighbors pairs连通}}{\#\text{ v-neighbors pairs}}$

$E[C(v)]=\sum_{d=0}^{d=n-1}E(C(v)\mid d_v=d)P(d_v=d)$ ↓组-对

$=\sum\frac{\binom{d}{2}p}{\binom{d}{2}}P(d_v=d)=p\sum_{d=0}^{d=n-1}P(d_v=d)=p$ ← 和为1

→ d 个neibor, 期望边数为 $C_d^2\cdot p$ ↙对间有边概率

无法解释现实高 clustering coefficient

Clustering Coefficient has phase transition

Average Path Length $\approx\frac{\ln|V|}{\ln d}$ ↙number of nodes ← average degree 总的来看小

X degree distribution

3. Small World Model 遇到困难: RG coefficient 太小, 而 regular network 路径太大

结合规则 lattice 的高聚类性和随机图的短路径

1. 先把环 Regular lattice

① 每个节点连最近 $d/2$ 个邻居 (previous $\frac{d}{2}$ + following $\frac{d}{2}$)

$\frac{1+\frac{n}{2d}}{2}=\frac{1}{2}+\frac{n}{2d}$

clustering coefficient $\approx\frac{3(d-2)}{4(d-1)}$

diameter $\frac{n}{d}$

long avg. shortest path length $\approx\frac{n}{2d}$

$d\times\frac{1+2+\cdots+\frac{n}{d}}{n-1}\approx d\times\frac{(1+\frac{n}{d})\times\frac{n}{d}}{2n}\approx\frac{n}{2d}$

d nodes 1 shortest path

把另一端移去其他node ↓ 这个和real graph 不一致

2 shortest path $\frac{1+\frac{n}{d}}{2}$

$\frac{n}{2d}$

2. 操作 rewriting 以概率 p 重连每条边 (显著降低 avg short path, 且仍未持高 cc)

仍不遵循 power law! 所有nodes degrees 相近

p 只要稍微增加一点, 路径长度就会迅速下降, 但 cc 下降缓慢

$C(p)=(1-p)^3C(0)$ ↙ $C(0)$ 是没 rewiring 的 cc

$C(0)=\frac{3(d-2)}{4(d-1)}$

给定 d_{avg}, p, n 就可以simulate small-world system

原有 triplet 三条边都没被 rewire 的概率

4. Preferential Attachment Model 优先依附模型

先前 challenge: Small World Model 无法产生 power law

解决思路：引入动态增长机制 和 richer 更 rich

1. 增长：从一个小型图开始，新节点不断加入

2. 优先依附：新节点连接到现有节点 v_i 的 p 与该节点 degree 成正比.

$P(v_i)=\frac{k_i}{\sum_j k_j}$ 加入社交网络时优先连 popular stars

添加时每个点 make $m \approx \frac{k}{n}$ connections（k: 边数, n: nodes数）

Power law ✓

Avg short path 小 ✓

CC：低（生成时未考虑传递性）X

	现实	Random graph	Small World Model	Preferential Attachment Model
Degree分布	P-L	Bino	X P-L	P-L
CC	High	Low (p)	High $\frac{3(d-2)}{4(d-1)}$	Low
Avg. shortest path	Small	Small $\frac{\ln\|V\|}{\ln d}$	Small $\frac{n}{2d}$	Small $C\sim\frac{(\ln(N))^2}{N}$

p 很小；$\ln d$：average degree；$\frac{n}{d}$：diameter

Adamic_adar_index

$\sum_{w\in N(u)\cap N(v)} \frac{1}{\log|N(w)|}$ ← 交集

Similarity

Holme kim Model

不仅进行优先连接，还会尝试与其邻居的邻居建立连接.

$P(v_i)=\frac{k_i}{\sum_j k_i}$ 先连到个 node m.

后以概率 P 连到其邻居 n（n: m）

形成 triplet

common neighbour and Centrality based Parameterized Algorithm (CCPA)

$CPA := \alpha|N(u)\cap N(v)| + (1-\alpha)\frac{N}{d_{uv}}$

d_{uv}：shortest distance between u and v

Regular Equivalence

$Gr(v_i,v_j)=\alpha\sum_{v_k\in N(v_i)} A_{ik}\cdot Gr(v_k,v_j)$

$Gr=(I-\alpha A^T)^{-1}$ $\alpha<\frac{1}{\lambda_{max}}$

$I-\alpha A^T$ 可逆 ⇔ 不可逆 ⇔ 特征值为0

设 $A^Tx=\lambda x$ $(I-\alpha\lambda)x$

$(I-\alpha A^T)x=0$

$\alpha=\frac{1}{\lambda}$ 时不可逆.

$\alpha>\frac{1}{\lambda}$ 不收敛

由 add_node( ) u→v. DiGraph

add_nodes_from (It)

add_edge( )

add_edges_from([ ],[ ],...) 或 str.

successors 继承者：A successor of node n is a node m that have there exits a directed edge from n to m

被指向者

G.degree incident 关联. ← 列表 / 字典

单个 node：→ int

多个 node：→ [(node, int)....]

nbunch：None 所有点

nx.Graph( )

↳ None / 其他 graph 对象 / edge list / adjacency dict

G[node] 返回 {neigh: { }, ...} G[1] = G.adj[1]

G.edges.data("att") ⇒ (u, v, wt)

↑ att

pos = nx.spring_layout(G, seed) att

nx.draw(G, pos, with_label, node_color, node_size, font_size, edge_color)

circular_layout 使合 flow chart

Week 8–9: Community Analysis Study Notes

1 Core Objective and Logical Evolution

The central goal of this chapter is: given a social network graph $G = (V, E)$, discover communities from the network structure. A community should have dense internal links and sparse external links. The chapter follows three major questions:

$$\text{Community Analysis} = \text{Detection} + \text{Evolution} + \text{Evaluation}.$$

That is, we first ask how to find communities, then how communities evolve over time, and finally how to evaluate whether the detected communities are good.

Social network $G = (V, E)$
$\Downarrow$
Implicit community discovery
$\Downarrow$
Static detection: member-based and group-based methods
$\Downarrow$
Dynamic detection: evolutionary clustering
$\Downarrow$
Evaluation: with or without ground truth

2 Basic Concepts of Social Media Communities

2.1 Definition [Important]

A basic social media community exists when like-minded users form links and interact with each other. Any community formation requires:

$$\text{community} = \{\text{at least two nodes}\} + \{\text{interactions with respect to a shared interest}\}.$$

Thus, a community is not merely a set of users; it is a set of users connected by interest-driven interaction.
There are two major types of groups:

Type	Definition	Membership known?
Explicit group	Formed by user subscriptions	Yes
Implicit group	Formed implicitly by interactions	No, must be mined

Examples of explicit groups include Facebook groups and Twitter lists. Examples of implicit groups include users who watch similar movies or users who make similar international phone calls. In different contexts, a community may also be called a group, cluster, cohesive subgroup, or module.

3 Why Detect Implicit Communities?

3.1 Recommendation [Important]

Individuals often form implicit groups based on interests. Detecting such groups is similar to detecting users with similar interests:

$$\text{similar interaction pattern} \Rightarrow \text{similar interest} \Rightarrow \text{recommendation}.$$

For example, movie rental sites may detect users watching similar movies and use the resulting implicit communities for recommendation.

3.2 Fraud and Anomaly Detection [Important]

A group provides a clearer global view of user interactions, whereas local individual behavior is often noisy and ad hoc. For example, fake review workers may control multiple accounts. These malicious accounts may link to each other and also connect to honest users to mimic normal behavior. When assigned a task, they may post fake reviews within a short time. By constructing a co-activity graph, community detection can reveal such fraudster groups.

3.3 Group-Level Behavior [Medium]

Some behaviors are only observable at the group level, not at the individual level. Individual behavior may fluctuate, but collective group behavior is often more robust.

3.4 Visualization and Privacy [Medium]

An implicit community can summarize a set of nodes in the whole network:

$$G = (V, E) \quad \longrightarrow \quad \{C_1, C_2, \ldots, C_K\}.$$

Instead of releasing individual-level data, one may release only community-level properties, reducing privacy risks.

4 Overall Structure of Community Analysis

- Community Analysis
- Community Detection: member-based methods, including similarity-based, degree/clique-based, and reachability-based methods; group-based methods, including minimum cut, ratio cut / normalized cut, spectral clustering, and hierarchical clustering.
- Community Evolution: dynamic network patterns and evolutionary clustering.
- Community Evaluation: evaluation with ground truth and evaluation without ground truth.

5 Community Detection

5.1 Definition [Important]

Community detection is the process of finding clusters of nodes with strong internal connections and weak connections between different clusters:

$$\text{strong internal connections} + \text{weak external connections}.$$

An ideal decomposition separates a large graph into completely disjoint communities with no interaction between communities. In practice, the task is to find a partition into communities that are maximally decoupled.

6 Member-Based Community Detection

Member-based community detection focuses on properties of nodes and usually assumes that nodes with similar characteristics belong to the same community.

Criterion	Assumption	Representative methods
Degree	Nodes with similar degree form communities	Clique
Reachability	Nearby nodes form communities	k-clique, k-club, k-clan
Similarity	Similar nodes form communities	Jaccard, cosine, k-means

6.1 Similarity-Based Community Detection [Important]

The intuition is:

$$\text{similar nodes} \Rightarrow \text{same community}.$$

Vertex similarity can be defined using either neighborhood similarity or node-attribute similarity. Then k-means or other similarity-based clustering methods can be applied.

6.1.1 Jaccard Similarity

$$\sigma_{\text{Jaccard}}(v_i, v_j) = \frac{|N(v_i) \cap N(v_j)|}{|N(v_i) \cup N(v_j)|}.$$

Here, v_i, v_j are two nodes; $N(v_i)$ is the neighbor set of v_i; $N(v_i) \cap N(v_j)$ is the common-neighbor set; and $N(v_i) \cup N(v_j)$ is the union of their neighbors. The larger the fraction of shared neighbors, the more similar the two nodes are.

6.1.2 Cosine Similarity

$$\sigma_{\text{cosine}}(v_i, v_j) = \frac{|N(v_i) \cap N(v_j)|}{\sqrt{|N(v_i)|\,|N(v_j)|}}.$$

If each node is represented by a neighbor indicator vector

$$\mathbf{x}_i[s] = \begin{cases} 1, & s \in N(v_i), \\ 0, & s \notin N(v_i), \end{cases}$$

then

$$\cos(v_i, v_j) = \frac{\mathbf{x}_i^\top \mathbf{x}_j}{\|\mathbf{x}_i\|_2 \|\mathbf{x}_j\|_2} = \frac{|N(v_i) \cap N(v_j)|}{\sqrt{|N(v_i)|\,|N(v_j)|}}.$$

For the slide example:

$$Jaccard(4,6) = \frac{|\{5\}|}{|\{1,3,4,5,6,7,8\}|} = \frac{1}{7},$$

$$cosine(4,6) = \frac{1}{\sqrt{4 \cdot 4}} = \frac{1}{4}.$$

6.2 Degree-Based Method: Clique [Important]

A clique is a complete subgraph in which every pair of nodes is adjacent. A clique of size k contains k nodes, and each node has internal degree at least $k-1$:

$$\text{size-}k \text{ clique} \Rightarrow \forall v \in C, \quad d_C(v) = k - 1.$$

Large cliques are naturally communities because their internal density is maximal:

$$\text{internal density} = 1.$$

However, clique-based detection has serious limitations: it is too strict, unstable to the removal of one edge, and computationally expensive for large cliques.

6.3 Finding a Clique of Size k [Medium]

If we want to find a clique of size k, every node in it must have degree at least $k-1$. Therefore:

$$d(v) < k - 1 \Rightarrow v \text{ cannot belong to a size-}k \text{ clique.}$$

A pruning procedure is:

Input: graph G, target clique size k.
Repeat: remove all nodes with degree $< k-1$.
Stop when a clique of size k is found or all nodes are removed.

This is useful because many nodes can be pruned in social networks, whose degree distributions often follow a power law.

6.4 Clique Percolation Method, CPM [Important]

The definition of a clique is too strict and unstable. CPM uses small cliques as cores or seeds and merges highly overlapping cliques into larger communities. It can detect overlapping communities.

Input:

$$k, \quad G = (V, E).$$

Algorithm:

1. Find all cliques of size k.
2. Construct a clique graph.
3. Two k-cliques are adjacent if they share $k-1$ nodes.
4. Connected components in the clique graph form communities.

Formally, two cliques C_a, C_b are adjacent if

$$|C_a \cap C_b| = k - 1.$$

The resulting community is

$$\mathcal{C} = \bigcup_{C_i \in \text{component}} C_i.$$

The intuition is that if two k-cliques share $k-1$ nodes, they differ by only one node and should belong to the same community.

Method	Advantage	Limitation
Clique	Very strong community definition	Too strict, fragile, expensive
CPM	Allows overlap and is more stable	Sensitive to k and still needs clique search

6.5 Reachability-Based Method: k-Clique [Important]

A k-clique is not the same as a clique of size k. A k-clique is a maximal subgraph in which the largest geodesic distance between any two nodes is at most k:

$$\max_{u,v \in C} d_G(u,v) \leq k.$$

Here, $d_G(u,v)$ is the shortest-path distance in the original graph G, and maximal means that no more nodes can be added while preserving the property.

Concept	Definition	Key condition
Clique of size k	A complete subgraph with k nodes	Every pair is directly connected
k-Clique	A maximal subgraph with diameter $\leq k$	Every pair is reachable within k hops

Therefore,

$$\text{clique of size } k \neq k\text{-clique.}$$

7 Group-Based Community Detection

Group-based methods use global network topology rather than only local node properties. The lecture discusses three goals:

Desired community property	Method
Balanced	Spectral clustering
Modular	Modularity maximization
Hierarchical	Hierarchical clustering

The main methodological evolution is:

$$\text{Community detection as cut} \Rightarrow \text{Minimum cut} \Rightarrow \text{Ratio/Normalized cut} \Rightarrow \text{Spectral clustering.}$$

8 From Minimum Cut to Ratio Cut and Normalized Cut

8.1 Community Detection as a Cut Problem [Important]

If most interactions are within communities and only a few interactions occur between communities, then community detection can be formulated as a cut problem. A cut partitions the vertex set into two disjoint sets:

$$V = P \cup \bar{P}, \qquad P \cap \bar{P} = \varnothing.$$

The cut size is

$$\text{cut}(P,\bar{P}) = |\{(u,v) \in E : u \in P,\ v \in \bar{P}\}|.$$

A good community cut should break few links:

$$\text{cut}(P,\bar{P}) \text{ small.}$$

8.2 Failure of Minimum Cut [Important]

Minimum cut solves

$$\min_P \text{cut}(P,\bar{P}).$$

However, it often returns imbalanced partitions, such as a singleton community. For example, if a low-degree node v is connected to the graph by only one edge, then

$$\text{cut}(\{v\}, V \setminus \{v\}) = 1.$$

This cut may be minimal, but it does not represent a meaningful community. The failure occurs because minimum cut only penalizes broken edges and ignores community size.

9 Ratio Cut [Important]

9.1 Two-Community Case

For a cut that divides the graph into P_i and $\bar{P}_i$, the ratio cut score is

$$RC(P_i) = \frac{1}{2}\left(\frac{\text{cut}(P_i, \bar{P}_i)}{|P_i|} + \frac{\text{cut}(P_i, \bar{P}_i)}{|\bar{P}_i|}\right).$$

Here, P_i is one community, $\bar{P}_i = V \setminus P_i$, $\text{cut}(P_i, \bar{P}_i)$ is the number of edges crossing the cut, and $|P_i|$ is the number of nodes in P_i.
A small ratio cut means both a small cut size and more balanced community sizes. If a singleton is cut out, then $|P_i| = 1$, so the denominator cannot dilute the cut cost.

9.2 K-Community Case

If the graph is divided into

$$\{P_1, P_2, \ldots, P_K\},$$

then

$$RC(P_1, \ldots, P_K) = \frac{1}{K}\sum_{i=1}^{K} \frac{\text{cut}(P_i, \bar{P}_i)}{|P_i|}.$$

10 Normalized Cut [Important]

Ratio cut normalizes by the number of nodes, whereas normalized cut normalizes by the total degree of a community.

10.1 Two-Community Case

$$NC(P_i) = \frac{1}{2}\left(\frac{\text{cut}(P_i, \bar{P}_i)}{\text{vol}(P_i)} + \frac{\text{cut}(P_i, \bar{P}_i)}{\text{vol}(\bar{P}_i)}\right),$$

where

$$\text{vol}(P_i) = \sum_{v \in P_i} d(v).$$

Here, $d(v)$ is the degree of node v, and $\text{vol}(P_i)$ is the total degree of community P_i.

10.2 K-Community Case

$$NC(P_1, \ldots, P_K) = \frac{1}{K}\sum_{i=1}^{K} \frac{\text{cut}(P_i, \bar{P}_i)}{\text{vol}(P_i)}.$$

Method	Denominator	Balance criterion
Ratio Cut	$\lvert P_i \rvert$	Balanced number of nodes
Normalized Cut	$\text{vol}(P_i)$	Balanced total degree

Normalized cut is often more appropriate for social networks with heavy-tailed or power-law degree distributions.

11 From Cut Objectives to Spectral Clustering [Most Important]

Enumerating all possible cuts is impractical because the number of possible partitions grows exponentially with the number of nodes. Spectral clustering reformulates ratio cut or normalized cut as a matrix optimization problem.

11.1 Membership Matrix

Assume there are n nodes and K communities. Define

$$X = [\mathbf{x}_1, \ldots, \mathbf{x}_K] \in \{0,1\}^{n \times K},$$

where

$$X_{ij} = \begin{cases} 1, & \text{node } i \text{ belongs to community } j, \\ 0, & \text{otherwise.} \end{cases}$$

The vector $\mathbf{x}_j$ is the indicator vector of community j. For example, if nodes $1, 2, 3, 4$ are in community 1 and nodes $5, 6, 7, 8, 9$ are in community 2, then

$$\mathbf{x}_1 = [1, 1, 1, 1, 0, 0, 0, 0, 0]^\top,$$
$$\mathbf{x}_2 = [0, 0, 0, 0, 1, 1, 1, 1, 1]^\top.$$

11.2 Adjacency Matrix and Degree Matrix

The adjacency matrix is

$$A_{st} = \begin{cases} 1, & (s,t) \in E, \\ 0, & (s,t) \notin E. \end{cases}$$

The degree matrix is

$$D = \operatorname{diag}(d_1, d_2, \ldots, d_n),$$

where

$$d_s = \sum_t A_{st}.$$

11.3 $\mathbf{x}_i^\top A \mathbf{x}_i$: Twice the Number of Internal Edges

$$\mathbf{x}_i^\top A \mathbf{x}_i = \sum_{s,t} X_{si} A_{st} X_{ti}.$$

The term contributes only when both s and t are in community i. In an undirected graph, each internal edge is counted twice, once as A_{st} and once as A_{ts}. Thus,

$$\mathbf{x}_i^\top A \mathbf{x}_i = 2 \times \#\{\text{edges inside } P_i\}.$$

正好是2倍
内部. degree也会算两次.

11.4 $\mathbf{x}_i^\top D \mathbf{x}_i$: Total Degree Inside the Community

$$\mathbf{x}_i^\top D \mathbf{x}_i = \sum_s X_{si}^2 D_{ss} = \sum_{s \in P_i} d_s = \operatorname{vol}(P_i).$$

11.5 Graph Laplacian

The graph Laplacian is

$$L = D - A.$$

Then

$$\mathbf{x}_i^\top L \mathbf{x}_i = \mathbf{x}_i^\top (D - A) \mathbf{x}_i = \mathbf{x}_i^\top D \mathbf{x}_i - \mathbf{x}_i^\top A \mathbf{x}_i.$$

This equals the cut size:

$$\mathbf{x}_i^\top L \mathbf{x}_i = \operatorname{cut}(P_i, \bar{P}_i).$$

The intuition is:

$$\text{total degree of nodes in } P_i = 2 \times \text{internal edges} + \text{external cut edges}.$$

Since $\mathbf{x}_i^\top A \mathbf{x}_i$ counts twice the internal edges, subtracting it from the total degree leaves only the external cut edges.

11.6 Matrix Form of Ratio Cut

Because

$$|P_i| = \mathbf{x}_i^\top \mathbf{x}_i,$$

we have

$$RC = \frac{1}{K} \sum_{i=1}^{K} \frac{\operatorname{cut}(P_i, \bar{P}_i)}{|P_i|} = \frac{1}{K} \sum_{i=1}^{K} \frac{\mathbf{x}_i^\top L \mathbf{x}_i}{\mathbf{x}_i^\top \mathbf{x}_i}.$$

Define the normalized indicator vector:

$$\hat{\mathbf{x}}_i = \frac{\mathbf{x}_i}{(\mathbf{x}_i^\top \mathbf{x}_i)^{1/2}}.$$

Then

$$\hat{\mathbf{x}}_i^\top \hat{\mathbf{x}}_i = 1, \qquad \hat{\mathbf{x}}_i^\top \hat{\mathbf{x}}_j = 0, \quad i \neq j,$$

because disjoint communities do not share nodes. Therefore,

$$\hat{X}^\top \hat{X} = I.$$

The ratio cut objective becomes

$$RC = \frac{1}{K}\sum_{i=1}^{K} \hat{\mathbf{x}}_i^\top L \hat{\mathbf{x}}_i = \frac{1}{K}\operatorname{Tr}(\hat{X}^\top L \hat{X}).$$

Thus the optimization problem is

$$\min_{\hat{X}} \operatorname{Tr}(\hat{X}^\top L \hat{X}) \quad \text{s.t.} \quad \hat{X}^\top \hat{X} = I.$$

11.7 Why Eigenvectors? [Most Important]

If the discrete constraint $\hat{X} \in \{0,1\}^{n\times K}$ is relaxed and $\hat{X}$ is allowed to be continuous, then the Rayleigh–Ritz theorem implies that the optimal solution is given by the eigenvectors of L corresponding to the smallest K eigenvalues.
Hence spectral clustering proceeds as:

1. Construct A and D.
2. Compute $L = D - A$.
3. Take eigenvectors of L with the smallest eigenvalues.
4. Use these eigenvectors as node embeddings.
5. Run k-means on the embeddings.

The first eigenvector usually corresponds to all nodes belonging to the same cluster and is therefore not useful for splitting. For two-way clustering, the second smallest eigenvector, called the Fiedler vector, is often crucial.

12 Hierarchical Clustering

Real communities may have hierarchical structure:

$$\text{University} \rightarrow \text{Colleges} \rightarrow \text{Departments}.$$

Hierarchical clustering allows network analysis at different resolutions.

Method	Direction	Idea
Divisive hierarchical clustering	Top-down	Repeatedly split communities
Agglomerative hierarchical clustering	Bottom-up	Repeatedly merge communities

12.1 Divisive Hierarchical Clustering [Important]

Divisive clustering starts from the whole graph and repeatedly removes weak or bridge-like links:

Start with the whole graph.
Find the weakest or most bridge-like edge.
Remove the edge.
Recompute edge strengths.
Repeat until the desired number of connected components is obtained.

Each connected component becomes a community.

12.2 Edge Betweenness [Important]

Edge betweenness measures the percentage of shortest paths that pass through an edge:

$$EB(e) = \sum_{s<t} \frac{\sigma_{st}(e)}{\sigma_{st}}.$$

Here, e is an edge; σ_{st} is the number of shortest paths from node s to node t; and $\sigma_{st}(e)$ is the number of those shortest paths that pass through edge e.
An edge with high betweenness tends to be a bridge between communities:

$$\text{high edge betweenness} \Rightarrow \text{bridge edge} \Rightarrow \text{remove} \Rightarrow \text{community separation}.$$

This is the intuition behind the Girvan–Newman style divisive clustering method.

12.3 Agglomerative Hierarchical Clustering [Medium]

Agglomerative clustering works in the opposite direction:

Initialize each node as a community.
Merge communities successively according to a criterion.
Stop when the desired number of communities is reached.

The merging criterion may be modularity increase or vertex similarity.

13 Community Evolution

Static community detection studies a network at one time point. Real networks are dynamic:

$$G_1, G_2, \ldots, G_t.$$

The main questions are: how does a network change over time, how does a community change over time, what properties remain stable, and what properties change?

13.1 Dynamic Network Patterns

13.1.1 Pattern 1: Decreasing Connection Probability with Distance ***[Important]***

The probability of a new connection decreases as graph distance increases:

$$P((u,v) \text{ forms a new edge}) \downarrow \quad \text{as} \quad d(u,v) \uparrow .$$

Two users with shorter distance are more likely to know each other or share similar interests.

13.1.2 Pattern 2: Triadic Closure ***[Important]***

Many new connections occur as triadic closures:

$$A - B, \quad B - C \Rightarrow A - C.$$

This reflects the principle "a friend of my friend is my friend" and leads to a high clustering coefficient.

13.1.3 Pattern 3: Graph Densification ***[Important]***

The density of a graph increases as the network grows:

$$density(G) = \frac{|E|}{\#\text{possible edges}}.$$

For an undirected simple graph,

$$\#\text{possible edges} = \frac{|V|(|V|-1)}{2}.$$

The densification law is

$$E(t) \propto V(t)^{\alpha}, \quad 1 \le \alpha \le 2.$$

Here, $E(t)$ and $V(t)$ are the numbers of edges and nodes at time t. If $\alpha = 1$, edge growth is linear and the average degree is approximately constant. If $\alpha = 2$, growth is quadratic and the graph approaches a clique. Real networks often satisfy $1 < \alpha < 2$, meaning that the number of edges grows faster than the number of nodes.

13.1.4 Pattern 4: Diameter Shrinking ***[Important]***

Network diameter often shrinks over time:

$$diameter(G_t) \downarrow .$$

This means that the small-world phenomenon becomes more obvious as the network grows. The reason is densification: edges grow faster than nodes, creating shortcuts that reduce long paths.

13.2 Community Evolution Types [Medium]

Communities may expand, shrink, or dissolve:

$$\text{expand}, \quad \text{shrink}, \quad \text{dissolve}.$$

They may also merge or split, although the slides mainly emphasize expansion, shrinkage, and dissolution.

14 Community Detection in Evolving Networks

14.1 Snapshot-Based Extension [Important]

A direct extension of static methods is:

1. Take snapshots $G_1, G_2, \ldots, G_t$.
2. Run a static community detection algorithm independently on each G_i.
3. Assign community memberships across time, for example by voting.

The problem is instability:

highly dynamic network $\Rightarrow$ frequently changing memberships $\Rightarrow$ unstable communities.

14.2 Evolutionary Clustering [Important]

Evolutionary clustering assumes that communities change smoothly over time. The objective considers two costs:

- Snapshot Cost (SC) and Temporal Cost (TC).

A general objective is

$$Cost_t = \alpha SC_t + (1-\alpha)TC_t, \qquad \alpha \in [0,1].$$

Here, SC_t measures how well the clustering fits the current snapshot G_t, while TC_t measures temporal consistency with previous communities. If α is large, the model emphasizes the current graph; if α is small, it emphasizes temporal smoothness.

Thus:

good evolving communities = fit current graph + remain stable over time.

15 Community Evaluation

Community evaluation can be conducted with or without ground truth:

Setting	Methods
With ground truth	Precision, recall, F-measure, purity
Without ground truth	Domain experts, user studies, modularity, internal/external links

16 Evaluation with Ground Truth

When ground truth is available, we know the correct community assignments. Explicit communities can be used as ground truth. The evaluation compares:

predicted communities vs. ground-truth communities.

16.1 Pairwise TP/TN/FP/FN [Most Important]

Community evaluation defines TP, TN, FP, and FN over pairs of nodes.

Type	Ground truth	Prediction	Meaning
TP	similar / same class	same community	correct
TN	dissimilar / different class	different communities	correct
FP	dissimilar / different class	same community	wrong
FN	similar / same class	different communities	wrong

For any node pair (u, v), let

$$y_{uv} = 1$$

mean that u and v are in the same ground-truth class, and let

$$\hat{y}_{uv} = 1$$

mean that the algorithm assigns u and v to the same community. Then

$$TP = |\{(u,v) : y_{uv} = 1, \hat{y}_{uv} = 1\}|,$$
$$TN = |\{(u,v) : y_{uv} = 0, \hat{y}_{uv} = 0\}|,$$
$$FP = |\{(u,v) : y_{uv} = 0, \hat{y}_{uv} = 1\}|,$$
$$FN = |\{(u,v) : y_{uv} = 1, \hat{y}_{uv} = 0\}|.$$

16.2 Precision [Important]

$$Precision = P = \frac{TP}{TP + FP}.$$

Precision measures how many node pairs assigned to the same community are truly similar:

high precision $\Rightarrow$ pure communities.

However, if every node is placed into a singleton community, FP becomes small, but recall becomes poor. Therefore, precision alone is insufficient.

16.3 Recall [Important]

$$Recall = R = \frac{TP}{TP + FN}.$$

Recall measures how many truly similar node pairs are successfully placed into the same community:

high recall $\Rightarrow$ few true groups are split apart.

However, if all nodes are placed into one community, FN becomes zero and recall becomes high, but precision becomes poor. Therefore, recall alone is insufficient.

16.4 F-Measure [Important]

The F-measure combines precision and recall using the harmonic mean:

$$F = 2\frac{P \cdot R}{P + R}.$$

The harmonic mean penalizes imbalance. If one of P or R is low, F remains low. For example, if

$$P = 0.428, \qquad R = 0.428,$$

then

$$F = 2\frac{0.428 \times 0.428}{0.428 + 0.428} = 0.428.$$

16.5 Purity [Important]

Purity assumes that the majority label in a detected community represents that community. If the algorithm detects K communities

$$(P_1, P_2, \ldots, P_K),$$

then

$$Purity = \frac{\sum_{i=1}^{K} \max_j |P_i \cap C_j|}{N}.$$

Here, P_i is the i-th detected community, C_j is the j-th ground-truth class, $|P_i \cap C_j|$ is the number of nodes in P_i with ground-truth label C_j, and N is the total number of nodes.
The intuition is:

$$Purity = \frac{\text{total number of majority-label instances in all detected communities}}{\text{total number of nodes}}.$$

For example:

$$Purity = \frac{6 + 5}{14} = 0.78,$$

and

$$Purity = \frac{4 + 4}{14} = 0.57.$$

Purity can be easily manipulated. If every node forms a singleton community, then every community has purity 1, so

$$Purity = 1.$$

This is clearly not a meaningful clustering. Therefore, purity should not be used alone.

17 Evaluation without Ground Truth

Without ground truth, possible evaluation methods include domain experts, user studies, comparing several algorithms, measuring modularity, and comparing the number of within-community links with the number of between-community links. The core principle remains:

$$\text{many internal links} \quad \text{and} \quad \text{few external links.}$$

18 Methodological Evolution Summary

Stage	Method	What it solves	Limitation
1	Explicit group	Membership is directly known	Only works for subscription-based groups
2	Similarity-based detection	Finds implicit groups using node or neighborhood similarity	Local; may ignore global topology
3	Clique	Finds extremely dense subgraphs	Too strict, unstable, expensive
4	CPM	Merges clique seeds and supports overlap	Sensitive to k, still requires clique search
5	k-Clique	Relaxes direct adjacency into short-path reachability	May include loosely connected nodes
6	Minimum cut	Uses global structure and cuts inter-community edges	Tends to produce singleton or tiny communities
7	Ratio/Normalized cut	Considers both cut size and balance	Discrete optimization is hard
8	Spectral clustering	Converts cut optimization into eigenvector problem	Requires K, depends on k-means
9	Hierarchical clustering	Captures multi-resolution community structure	Sensitive to stopping rule and computational cost
10	Evolutionary clustering	Handles dynamic networks with temporal smoothness	Requires balancing snapshot and temporal costs

19 Final Exam Priority

Highest Priority

1. Community definition: strong internal connections and weak external connections.
2. Explicit communities versus implicit communities.
3. Member-based versus group-based detection.
4. Jaccard similarity and cosine similarity formulas.
5. Clique versus k-clique.
6. CPM algorithm and overlapping community detection.
7. Why minimum cut fails.
8. Ratio cut and normalized cut formulas and intuition.
9. Laplacian $L = D - A$ and the identity $\mathbf{x}^\top L\mathbf{x} = \text{cut}(P, \bar{P})$.
10. Spectral clustering optimization and eigenvector solution.
11. Edge betweenness and divisive clustering.
12. Dynamic patterns: triadic closure, densification, shrinking diameter.
13. Precision, recall, F-measure, purity, and their limitations.

Medium Priority

1. Zachary karate club example.
2. Scientists' collaboration network example.

3. Real-world examples of hierarchical communities.
4. Trade-off between SC and TC in evolutionary clustering.
5. Evaluation without ground truth.

Lower Priority

1. Facebook group and Twitter list screenshots.
2. Specific numerical edge-betweenness values in examples.
3. Detailed clique lists in each practice example, unless computational exercises are emphasized.

Week 10 Review Notes: Label Propagation and Node Embeddings

1 Week 10: From Label Propagation to Node Embeddings

1.1 Overall Theme

The central objective of this week is to understand how graph structure can be used to infer node-level information or to learn reusable node representations. Given a graph

$$G = (V, E),$$

we want to exploit structural notions such as neighborhood, similarity, and co-occurrence to either predict missing node labels or learn low-dimensional embeddings for downstream tasks.

There are two main methodological lines:

$$\text{partial node labels} + \text{graph structure} \Rightarrow \text{labels of unlabeled nodes},$$

which corresponds to **transductive / semi-supervised node classification**, represented by **Label Propagation**; and

$$\text{graph structure} \Rightarrow \text{node embeddings} \Rightarrow \text{downstream tasks},$$

which corresponds to **unsupervised / self-supervised graph representation learning**, represented by **random-walk-based node embeddings** such as DeepWalk and node2vec.

1.2 Transductive Node Classification

1.2.1 Problem Definition [Importance: ⋆⋆⋆⋆⋆]

Let

$$G = (V, E)$$

be a graph, where

$$V = \{1, 2, \ldots, n\}$$

is the node set, and $n = |V|$ is the number of nodes. The edge set is

$$E \subseteq V \times V.$$

For an undirected graph, if $(u, v) \in E$, then $(v, u) \in E$.

The adjacency matrix is

$$A \in \mathbb{R}^{n \times n},$$

where

$$A_{uv} = \begin{cases} 1, & \text{if } (u, v) \in E, \\ 0, & \text{otherwise.} \end{cases}$$

For a weighted graph, A_{uv} may represent the edge weight between nodes u and v.

The node-label vector is

$$Y = \{0, 1\}^n,$$

where

$$Y_v = 1$$

means node v belongs to class 1, and

$$Y_v = 0$$

means node v belongs to class 0. In semi-supervised node classification, only part of the node labels are known. We decompose the node set as

$$V = V_L \cup V_U, \quad \text{labeled} \quad \text{unknown}$$

where V_L is the set of labeled nodes and V_U is the set of unlabeled nodes. The goal is

$$\text{predict } Y_v, \quad \forall v \in V_U.$$

1.2.2 Core Assumption: Correlation in Networks [Importance: ⋆⋆⋆⋆⋆]

Node classification is possible because node behaviors or labels are often correlated across network links:

nearby nodes tend to have similar labels.

There are two major explanations.

Homophily. Homophily means that similar nodes are more likely to connect:

similarity in individual characteristics ⇒ social connection.

This is often summarized as "birds of a feather flock together." For example, researchers working in the same research area are more likely to collaborate or interact.

Influence. Influence means that connected nodes may become similar because they affect each other:

social connection ⇒ similarity in individual characteristics.

For example, a person may recommend music to friends, and those friends may gradually develop similar musical preferences.

The distinction is important:

homophily: similar first, then connected;

influence: connected first, then similar.

Both mechanisms lead to the same observable phenomenon:

connected nodes often have correlated labels.

1.3 Method A: Label Propagation

1.3.1 Motivation [Importance: ⋆⋆⋆⋆⋆]

If neighboring nodes tend to have similar labels, then the label of an unknown node can be inferred from its neighbors. The core idea is:

the class probability of a node ⇒ a weighted average of the class probabilities of its neighbors.

Let

$$P^{(t)}(Y_v = c)$$

denote the probability that node v belongs to class c at iteration t. In the binary case, the slides often write

$$P(Y_v) = P(Y_v = 1),$$

namely the probability that node v belongs to class 1.

1.3.2 Initialization [Importance: ⋆⋆⋆⋆⋆]

For labeled nodes $v \in V_L$, initialize and fix their probabilities using the ground-truth label:

$$P^{(0)}(Y_v = 1) = Y_v^*,$$

where Y_v^* denotes the true label. Thus,

$$Y_v^* = 1 \Rightarrow P^{(0)}(Y_v = 1) = 1,$$

and

$$Y_v^* = 0 \Rightarrow P^{(0)}(Y_v = 1) = 0.$$

For unlabeled nodes $v \in V_U$, initialize with uncertainty:

$$P^{(0)}(Y_v = 1) = 0.5.$$

1.3.3 *Iterative Update Rule [Importance: ⋆⋆⋆⋆⋆]*

For each unlabeled node $v \in V_U$, update its class probability by averaging over neighbors:

$$P^{(t)}(Y_v = c) = \frac{\sum_{u \in N(v)} A_{vu} P^{(t-1)}(Y_u = c)}{\sum_{u \in N(v)} A_{vu}}.$$

Here,

$$N(v) = \{u \in V : (u, v) \in E\}$$

is the neighbor set of node v, A_{vu} is the edge weight between v and u, $P^{(t)}(Y_v = c)$ is the updated probability at iteration t, and $P^{(t-1)}(Y_u = c)$ is the previous-iteration probability of neighbor u.

For an unweighted graph, this reduces to

$$P^{(t)}(Y_v = c) = \frac{1}{|N(v)|} \sum_{u \in N(v)} P^{(t-1)}(Y_u = c).$$

The intuition is simple: a node receives votes from its neighbors, and the probability is the average vote.

1.3.4 *Synchronous Update [Importance: ⋆⋆⋆⋆]*

The update uses previous-iteration scores:

$$P^{(t)}(Y_v) \leftarrow \text{average of } P^{(t-1)}(Y_u).$$

This is a synchronous update. If the algorithm instead used values already updated in the same iteration, the result would depend on the order in which nodes are updated. Synchronous updating gives a clearer and more reproducible algorithm.

1.3.5 *Example Computations [Importance: ⋆⋆⋆⋆]*

For node 3,

$$N(3) = \{1, 2, 4\}.$$

At initialization,

$$P(Y_1) = 0, \quad P(Y_2) = 0, \quad P(Y_4) = 0.5.$$

Therefore, after the first iteration,

$$P^{(1)}(Y_3) = \frac{0 + 0 + 0.5}{3} = 0.1667 \approx 0.17.$$

For node 4,

$$N(4) = \{1, 3, 5, 6\}.$$

Using previous-iteration values,

$$P(Y_1) = 0, \quad P(Y_3) = 0.5, \quad P(Y_5) = 0.5, \quad P(Y_6) = 1.$$

Thus,

$$P^{(1)}(Y_4) = \frac{0 + 0.5 + 0.5 + 1}{4} = 0.5.$$

For node 5,

$$N(5) = \{4, 6, 7, 8\}.$$

Using previous-iteration values,

$$P(Y_4) = 0.5, \quad P(Y_6) = 1, \quad P(Y_7) = 1, \quad P(Y_8) = 0.5.$$

Thus,

$$P^{(1)}(Y_5) = \frac{0.5 + 1 + 1 + 0.5}{4} = 0.75.$$

After about four iterations in the slide example, the probabilities stabilize approximately as

$$P(Y_4) \approx 0.51, \quad P(Y_5) \approx 0.86, \quad P(Y_8) \approx 0.95, \quad P(Y_9) = 1, \quad P(Y_3) \approx 0.17.$$

Using threshold 0.5,

$$P(Y_v) > 0.5 \Rightarrow \text{class } 1,$$
$$P(Y_v) < 0.5 \Rightarrow \text{class } 0.$$

Therefore,

$$4, 5, 8, 9 \in \text{class } 1,$$

and

$$3 \in \text{class } 0.$$

1.3.6 Convergence Criterion [Importance: ⋆⋆⋆⋆⋆]

The stopping condition is

$$\left|P^{(t)}(Y_v) - P^{(t-1)}(Y_v)\right| \leq \epsilon, \quad \forall v \in V.$$

Here, t is the iteration number, $P^{(t)}(Y_v)$ is the probability after iteration t, $P^{(t-1)}(Y_v)$ is the previous probability, and ϵ is the convergence threshold. The intuition is that once all node probabilities almost stop changing, further propagation no longer adds meaningful information.

1.3.7 Limitations of Label Propagation [Importance: ⋆⋆⋆⋆⋆]

Label Propagation is simple and intuitive, but it has several limitations. First, convergence may be slow and is not always guaranteed. Second, it only uses graph structure and ignores node attributes f_v, where f_v is the feature vector of node v. Third, it is task-specific: it directly solves the current label prediction problem rather than learning a reusable representation. Fourth, it relies heavily on homophily. If the graph is heterophilous, meaning adjacent nodes tend to have different labels, label propagation may fail.

1.4 Methodological Evolution: Why Node Embeddings?

Label Propagation follows the principle

labels propagate over edges.

Its weakness is that it is tied to one label-prediction task and mainly exploits local neighborhood information. Node embedding changes the objective:

do not directly propagate labels; first learn node representations.

The goal is to learn a mapping

$$f : u \mapsto \mathbb{R}^d,$$

or equivalently

$$f(u) = z_u,$$

where $u \in V$, $z_u \in \mathbb{R}^d$ is the embedding vector of node u, and d is the embedding dimension, usually much smaller than n.

This gives the pipeline

graph structure ⇒ node vectors ⇒ classification, link prediction, clustering, anomaly detection, etc.

Thus, node embedding is a form of task-independent representation learning.

1.5 Graph Representation Learning and Node Embedding

1.5.1 Task Definition [Importance: ⋆⋆⋆⋆⋆]

The objective is to map each node to a low-dimensional vector:

$$f : V \to \mathbb{R}^d.$$

For node u,

$$f(u) = z_u,$$

where $z_u \in \mathbb{R}^d$. The core requirement is

similarity in embedding space ≈ similarity in the graph.

A common similarity measure is the dot product:

$$z_u^\top z_v.$$

If two nodes are similar in the graph, then $z_u^\top z_v$ should be large; if they are dissimilar, $z_u^\top z_v$ should be small.

1.5.2 Shallow Encoding [Importance: ⋆⋆⋆⋆]

The simplest encoder is an embedding lookup table. Let

$$Z \in \mathbb{R}^{d \times |V|}$$

be the embedding matrix. Each column corresponds to one node:

$$Z[:, u] = z_u.$$

Therefore,

$$\mathrm{ENC}(u) = Z[:, u] = z_u.$$

This method does not use node features or a deep neural encoder. Each node is directly assigned a unique trainable vector. Representative methods include

DeepWalk, node2vec.

It is called shallow because there is no multi-layer message passing and no feature-based generation of node embeddings.

1.6 Random-Walk Embeddings

1.6.1 Core Idea [Importance: ⋆⋆⋆⋆⋆]

The main idea of random-walk embeddings is:

if a random walk starting from u frequently visits v, then u and v are similar.

Let

$$P_R(v|u)$$

denote the probability that a random walk (strategy R) starting from node u, visits node v. The embedding objective is to make

$$z_u^\top z_v \approx P_R(v|u).$$

More precisely, a larger dot product should imply that v is more likely to appear in the random-walk neighborhood of u. The slide intuition is

$$z_u^\top z_v \approx \Pr(u \text{ and } v \text{ co-occur on a random walk}).$$

1.6.2 Random Walk Definition [Importance: ⋆⋆⋆⋆]

Given a graph and a starting node u, a random walk repeatedly selects a random neighbor of the current node and moves to it. This produces a sequence

$$(u = v_0, v_1, v_2, \ldots, v_L),$$

where L is the walk length and

$$v_{t+1} \in N(v_t).$$

1.6.3 Random-Walk Neighborhood [Importance: ⋆⋆⋆⋆⋆]

For each node u, a random-walk strategy R defines a neighborhood

通过策略R得到 $N_R(u)$.

Here, $N_R(u)$ is the multiset of nodes visited by random walks starting from u. It is a multiset rather than an ordinary set because nodes may appear multiple times; repeated occurrence captures higher visitation frequency.

可能有重复.

1.7 Optimization Objective for Random-Walk Embeddings

1.7.1 Maximum Likelihood Objective [Importance: ⋆⋆⋆⋆⋆]

Given

$$G = (V, E),$$

we learn

$$f : u \mapsto \mathbb{R}^d, \quad f(u) = z_u.$$

The embedding z_u should predict the random-walk neighborhood $N_R(u)$. The maximum likelihood objective is

$$\arg\max_Z \sum_{u \in V} \log P(N_R(u) \mid z_u).$$

Here, Z is the matrix of all node embeddings; $u \in V$ ranges over all nodes, z_u is the embedding of node u, $N_R(u)$ is the random-walk neighborhood of u, and $P(N_R(u) \mid z_u)$ is the probability of observing this neighborhood given z_u. A common conditional-independence assumption gives

$$P(N_R(u) \mid z_u) = \prod_{v \in N_R(u)} P(v \mid z_u).$$

Taking logarithms,

$$\log P(N_R(u) \mid z_u) = \sum_{v \in N_R(u)} \log P(v \mid z_u).$$

Therefore,

$$\arg\max_Z \sum_{u \in V} \sum_{v \in N_R(u)} \log P(v \mid z_u).$$

Equivalently, minimize the negative log-likelihood:

$$\arg\min_Z \mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log P(v \mid z_u).$$

1.7.2 Softmax Parameterization [Importance: ⋆⋆⋆⋆⋆]

The conditional probability is parameterized by softmax:

$$P(v \mid z_u) = \frac{\exp(z_u^\top z_v)}{\sum_{n \in V} \exp(z_u^\top z_n)}.$$

Here, $P(v \mid z_u)$ is the model-predicted probability that node v appears in the random-walk context of node u, $z_u^\top z_v$ is the embedding similarity between u and v, $\exp(z_u^\top z_v)$ converts the similarity into a positive score, and

$$\sum_{n \in V} \exp(z_u^\top z_n)$$

is the normalization term over all nodes.

Softmax is used because the model is predicting which node, among all nodes, is likely to appear in the random-walk neighborhood:

$$\sum_{v \in V} P(v \mid z_u) = 1.$$

1.7.3 Full Loss Function [Importance: ⋆⋆⋆⋆⋆]

Substituting the softmax into the negative log-likelihood gives

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(z_u^\top z_v)}{\sum_{n \in V} \exp(z_u^\top z_n)}\right).$$

This is one of the central formulas of the lecture. For every center node u, and every context node v appearing in its random-walk neighborhood, the model assigns high probability to v. If $z_u^\top z_v$ is large, then

$$\exp(z_u^\top z_v)$$

is large, so

$$P(v \mid z_u)$$

is large and the loss becomes small. If $z_u^\top z_v$ is small, then the model assigns a low probability to a true context node, increasing the loss.

1.8 Softmax Bottleneck and Negative Sampling

1.8.1 Why Naive Softmax Is Expensive [Importance: ⋆⋆⋆⋆⋆]

The bottleneck is the denominator

$$\sum_{n \in V} \exp(z_u^\top z_n).$$

For every positive pair (u, v), the model must sum over all nodes $n \in V$. When $|V|$ is large, this is computationally expensive. The nested summation leads to approximately

$$O(|V|^2)$$

complexity. Hence, full softmax does not scale well to large graphs.

1.8.2 Core Idea of Negative Sampling [Importance: ⋆⋆⋆⋆⋆]

Negative sampling avoids normalizing over all nodes. Instead, for each positive pair (u, v), sample only k negative nodes:

$$n_i \sim P_V, \quad i = 1, \ldots, k.$$

The positive pair is

$$(u, v), \quad v \in N_R(u),$$

meaning that v truly appeared in the random-walk context of u. The negative samples n_i are drawn from a random node distribution P_V, often with probability proportional to node degree.

1.8.3 Sigmoid Function [Importance: ⋆⋆⋆⋆]

Negative sampling uses the sigmoid function:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}.$$

Here, $x \in \mathbb{R}$, and

$$\sigma(x) \in (0, 1).$$

If x is large, then

$$\sigma(x) \approx 1.$$

If x is very negative, then

$$\sigma(x) \approx 0.$$

1.8.4 Negative Sampling Objective [Importance: ⋆⋆⋆⋆⋆]

The original softmax log-probability

$$\log\left(\frac{\exp(z_u^\top z_v)}{\sum_{n\in V}\exp(z_u^\top z_n)}\right)$$

is approximated by

$$\log\sigma(z_u^\top z_v)+\sum_{i=1}^{k}\log\sigma(-z_u^\top z_{n_i}),\quad n_i\sim P_V.$$

正样本 负样本

The positive term

$$\log\sigma(z_u^\top z_v)$$

encourages true co-occurring nodes (u,v) to have large dot product. If

$$z_u^\top z_v\to+\infty,$$

要最大化目标.

then

$$\sigma(z_u^\top z_v)\to 1,$$

and

$$\log\sigma(z_u^\top z_v)\to 0.$$

The negative term

$$\sum_{i=1}^{k}\log\sigma(-z_u^\top z_{n_i})$$

encourages u and negative nodes n_i to have small similarity. If

$$z_u^\top z_{n_i}\to-\infty,$$

也

then

$$-z_u^\top z_{n_i}\to+\infty,$$

and

$$\sigma(-z_u^\top z_{n_i})\to 1.$$

也能最大化目标.

Thus, the model pulls positive pairs together and pushes negative pairs apart.

1.8.5 Loss-Function View [Importance: ⋆⋆⋆⋆⋆]

For one positive pair (u,v), the minimization loss is

$$\mathcal{L}_{u,v}=-\log\sigma(z_u^\top z_v)-\sum_{i=1}^{k}\log\sigma(-z_u^\top z_{n_i}).$$

The total loss is

$$\mathcal{L}=\sum_{u\in V}\sum_{v\in N_R(u)}\left[-\log\sigma(z_u^\top z_v)-\sum_{i=1}^{k}\log\sigma(-z_u^\top z_{n_i})\right].$$

This is more scalable than full softmax because each positive pair requires only

$$1+k$$

个postive pair

dot products instead of $|V|$ dot products.

1.8.6 Choice of k [Importance: ⋆⋆⋆⋆]

There are two considerations. A larger k gives more robust estimates:

$$k\uparrow\Rightarrow\text{more robust estimates.}$$

However, a larger k also places more emphasis on negative events:

$$k\uparrow\Rightarrow\text{higher bias toward negative samples.}$$

In practice,

$$k=5\sim 20.$$

1.9 Methodological Evolution Summary

Method	Input	What Is Learned	Strength	Limitation
Label Propagation	Graph structure + few labels	Label probabilities of unlabeled nodes	Simple and intuitive; suitable for semi-supervised node classification	Slow or non-guaranteed convergence; ignores node attributes; task-specific
Node Embedding	Graph structure	Node vector z_u	Task-independent; useful for classification, link prediction, clustering, anomaly detection	Shallow embeddings cannot naturally generalize to new nodes; does not directly use node attributes
Random-Walk Embedding	Graph structure + random walks	Embeddings preserving random-walk co-occurrence	Captures higher-order neighborhood information; avoids considering all node pairs during sampling	Full softmax is expensive
Negative Sampling	Positive pairs + sampled negative nodes	Efficient approximation of random-walk co-occurrence objective	Scalable to large graphs	Approximate objective; depends on negative sampling distribution and k

1.10 Logical Evolution Tree

Graph-based learning

$\Downarrow$

Node labels are correlated over edges

$\Downarrow$

Label Propagation

$\Downarrow$

Problem: task-specific, slow convergence, no node attributes

$\Downarrow$

Learn reusable node representations

$\Downarrow$

Node Embedding: $f(u) = z_u \in \mathbb{R}^d$

$\Downarrow$

Define graph similarity via random walks

$\Downarrow$

Maximize $P(N_R(u) \mid z_u)$

$\Downarrow$

Softmax objective

$\Downarrow$

Problem: denominator sums over all nodes

$\Downarrow$

Negative Sampling

$\Downarrow$

Efficient approximation with k sampled negative nodes.

1.11 Exam-Focused Formula Summary

$$P^{(t)}(Y_v = c) = \frac{\sum_{u \in N(v)} A_{vu} P^{(t-1)}(Y_u = c)}{\sum_{u \in N(v)} A_{vu}}.$$

$$\left| P^{(t)}(Y_v) - P^{(t-1)}(Y_v) \right| \leq \epsilon.$$

$$f : u \mapsto \mathbb{R}^d, \quad f(u) = z_u.$$

$$\arg\max_Z \sum_{u \in V} \log P(N_R(u) \mid z_u).$$

$$\arg\min_Z \mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log P(v \mid z_u).$$

$$P(v \mid z_u) = \frac{\exp(z_u^\top z_v)}{\sum_{n \in V} \exp(z_u^\top z_n)}.$$

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log \left(\frac{\exp(z_u^\top z_v)}{\sum_{n \in V} \exp(z_u^\top z_n)} \right).$$

$$\sigma(x) = \frac{1}{1 + \exp(-x)}.$$

$$\log \left(\frac{\exp(z_u^\top z_v)}{\sum_{n \in V} \exp(z_u^\top z_n)} \right) \approx \log \sigma(z_u^\top z_v) + \sum_{i=1}^{k} \log \sigma(-z_u^\top z_{n_i}).$$

$$\mathcal{L}_{u,v} = -\log \sigma(z_u^\top z_v) - \sum_{i=1}^{k} \log \sigma(-z_u^\top z_{n_i}).$$

1.12 One-Sentence Summary

Label Propagation propagates labels along graph edges; Node Embedding compresses graph structure into vectors; Random Walk defines which nodes are structurally similar; Softmax turns similarity into probability; Negative Sampling makes this objective computationally feasible on large graphs.

Week 11 Review Notes: From Graph Convolutions to Agent Skills

1 Overview: Two Evolutionary Lines

This week contains two independent but structurally similar technical lines.

First, **graph representation learning**: from naively feeding adjacency matrices into neural networks to defining deep graph models through neighborhood aggregation. The core goal is to learn embeddings for nodes, subgraphs, or entire graphs when graphs have no regular grid structure, arbitrary node orderings, and variable sizes.

Second, **LLM agents**: from language models that only generate text to structured tool use, MCP, skills, and sub-agents. The core goal is to let a pure text model safely and scalably interact with the external world through a host-executed protocol.

Graph learning: fixed matrix input $\rightarrow$ shared neighborhood computation

LLM: pure text output $\rightarrow$ structured external action through tools

2 Part I: Deep Graph Encoders and GCNs

2.1 Core Problem: Why Not Directly Feed Graphs into a DNN? [Very Important]

A graph is represented as

$$G = (V, E),$$

where V is the vertex set and E is the edge set. Its adjacency matrix is

$$A \in \{0,1\}^{|V|\times|V|},$$

where $A_{uv} = 1$ means that nodes u and v are connected, and $A_{uv} = 0$ otherwise. The node feature matrix is

$$X \in \mathbb{R}^{m\times|V|},$$

where the v-th column $x_v \in \mathbb{R}^m$ is the input feature vector of node v. If no meaningful node features exist, one may use one-hot indicator vectors.

A naive approach is to concatenate the adjacency matrix A and feature matrix X, then feed the result into a standard deep neural network. This fails for three fundamental reasons.

First, the number of parameters depends on the number of nodes $N = |V|$, so the model does not naturally generalize across graphs of different sizes. Second, standard DNNs assume a fixed input dimension, whereas real-world graphs such as social networks, biological networks, web graphs, and knowledge graphs are dynamic and variable-sized. Third, and most importantly, graphs do not have a canonical node ordering. If the same graph is relabeled by permuting node indices, its semantics should remain unchanged, but its adjacency matrix changes. A standard DNN is not permutation invariant or permutation equivariant.

Therefore, the goal of graph neural networks is not to flatten a graph into a fixed matrix, but to design a shared local computation rule that respects graph structure.

2.2 From CNNs to Graph Convolutions [Very Important]

CNNs work on images because images lie on regular lattices. For a 3×3 filter, every pixel has a fixed local neighborhood pattern. The essential operation of a convolutional layer is

transform neighbor information + combine it.

Graphs preserve the idea of local neighborhoods but destroy the assumptions of regular lattices: nodes have variable degrees, neighbors have no natural order, and local subgraphs may have different structures. Therefore, graph convolution attempts to generalize convolution from images to graphs.

The methodological evolution is

CNN on regular grids $\rightarrow$ graph convolution with canonical ordering

$\rightarrow$ message passing / neighborhood aggregation $\rightarrow$ learnable aggregation such as GraphSAGE and GAT.

Early graph convolution methods tried to impose a canonical ordering on each local subgraph and then apply CNN-like filters. The bottleneck is that canonical node ordering is difficult to define for arbitrary graphs. GCN-style models avoid this issue by using permutation-invariant aggregation functions such as mean, sum, max, or attention.

2.3 GCN Core Idea: A Node's Neighborhood Defines a Computation Graph [Very Important]

The key idea is:

A node's neighborhood defines its computation graph.

For a target node v, its representation is not computed independently. Instead, it is produced by recursively propagating and aggregating information from its K-hop neighborhood. The layer-0 embedding is the raw input feature:

$$h_v^0 = x_v.$$

Here h_v^0 is the hidden representation of node v at layer 0, and x_v is its input feature vector. After stacking K layers, the final node embedding is

$$z_v = h_v^K.$$

Thus, one GCN layer aggregates 1-hop information, two layers aggregate 2-hop information, and in general K layers allow node v to receive information from its K-hop neighborhood. The depth K controls the graph receptive field.

2.4 Basic GCN / Deep Encoder Formula [Very Important]

The basic neighborhood aggregation formula is

$$h_v^k = \sigma\left(W_k \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + B_k h_v^{k-1}\right), \qquad \forall k \in \{1, \ldots, K\},$$

with

$$h_v^0 = x_v, \qquad z_v = h_v^K.$$

The symbols are defined as follows. Node v is the target node. Node u is one of its neighbors. $N(v)$ denotes the neighbor set of v, and $|N(v)|$ is the number of neighbors. h_v^k is the embedding of node v at layer k. h_u^{k-1} is the previous-layer embedding of neighbor u. W_k is the trainable weight matrix for transforming aggregated neighbor messages at layer k. B_k is the trainable weight matrix for transforming the previous representation of the target node itself. $\sigma(\cdot)$ is a nonlinear activation function such as ReLU. z_v is the final embedding after K layers.

The formula can be decomposed into three steps. First, average neighbor messages:

$$m_v^k = \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|}.$$

Second, combine neighbor information and self-information:

$$a_v^k = W_k m_v^k + B_k h_v^{k-1}.$$

Third, apply a nonlinearity:

$$h_v^k = \sigma(a_v^k).$$

The intuition is that the new state of node v is determined by both the average opinion of its neighbors and its own previous state. The neighbor term enables information propagation over the graph, while the self-term prevents the node from losing its own identity and features.

2.5 Why Shared Parameters Enable Inductive Generalization [Very Important]

The crucial difference between a GCN and a naive DNN is parameter sharing. The same W_k and B_k are used for all nodes at layer k. Therefore, the computation graph of node A and the computation graph of node B use the same aggregation rule.

This means the model learns a function

$$f_\theta(v, X, A) \mapsto z_v,$$

where

$$\theta = \{W_k, B_k\}_{k=1}^K$$

is shared across nodes. Since the model does not learn a separate embedding table tied to fixed node identities, it can generate embeddings for unseen nodes or even entirely unseen graphs, as long as node features and neighborhoods are available.

This is the key difference between **inductive** and **transductive** node embedding. Traditional methods such as DeepWalk or node2vec often learn one embedding vector per training node. When a new node arrives, its embedding is not directly available. In contrast, GCN/GraphSAGE-style models learn an embedding function, so new embeddings can be computed on the fly.

2.6 Training: Unsupervised and Supervised Objectives [Important]

After obtaining embeddings, one must define a loss function.

2.6.1 Unsupervised Training

Unsupervised training uses only graph structure. The central principle is

similar nodes should have similar embeddings.

Similarity can be defined by random walks, graph factorization, or graph proximity. A typical random-walk skip-gram objective is

$$\mathcal{L} = -\sum_{v \in V} \sum_{u \in C(v)} \log p(u|v),$$

where $C(v)$ is the context set of node v obtained from random walks, and

$$p(u|v) = \frac{\exp(z_u^\top z_v)}{\sum_{n \in V} \exp(z_n^\top z_v)}.$$

Here $p(u|v)$ is the probability that node u appears in the context of node v, as predicted from their embeddings. In practice, negative sampling is often used:

$$\mathcal{L} = -\sum_{(v,u) \in \mathcal{D}} \log \sigma(z_v^\top z_u) - \sum_{(v,n) \in \mathcal{D}^-} \log \sigma(-z_v^\top z_n),$$

where $\mathcal{D}$ is the set of positive node pairs, $\mathcal{D}^-$ is the set of negative node pairs, and $\sigma(\cdot)$ is the sigmoid function.

2.6.2 Supervised Training

For supervised node classification, let y_v be the label of node v. A classifier is applied to the final embedding:

$$\hat{y}_v = \text{softmax}(W_{\text{out}} z_v + b_{\text{out}}).$$

The cross-entropy loss is

$$\mathcal{L} = -\sum_{v \in V_{\text{train}}} \sum_{c=1}^{C} \mathbf{1}[y_v = c] \log \hat{y}_{v,c}.$$

Here V_{train} is the training node set, C is the number of classes, and $\hat{y}_{v,c}$ is the predicted probability that node v belongs to class c. A typical example is drug classification in a drug-drug interaction network, where the model predicts whether a drug is safe or toxic.

2.7 From GCN to GraphSAGE: Why Is Mean Aggregation Not Enough? [Very Important]

In the basic GCN, the aggregation weight is

$$\alpha_{vu} = \frac{1}{|N(v)|},$$

which means all neighbors of node v contribute equally. This is a strong and often unrealistic assumption, because in real graphs different neighbors may have different semantic importance.

GraphSAGE generalizes neighborhood aggregation:

$$h_v^k = \sigma\left(W^k \cdot \text{CONCAT}\left(h_v^{k-1}, \text{AGGREGATE}_k\left(\{h_u^{k-1} : u \in N(v)\}\right)\right)\right).$$

Here $\text{AGGREGATE}_k(\cdot)$ can be mean aggregation, LSTM aggregation, pooling aggregation, or another permutation-invariant function.

GraphSAGE addresses two limitations of basic GCNs. First, mean aggregation may be too weak to capture complex neighborhood information. Second, for large dynamic graphs, it is expensive to aggregate over all neighbors, so GraphSAGE commonly uses neighbor sampling to support scalable inductive learning.

2.8 From GraphSAGE to GAT: Why Use Attention? [Very Important]

The remaining issue is that neighbor weights are still fixed or manually specified. Graph Attention Networks introduce learnable attention weights so that the model can assign different importances to different neighbors.
The basic graph convolution uses

$$\alpha_{vu} = \frac{1}{|N(v)|}.$$

GAT instead computes

$$h_v^k = \sigma\left(\sum_{u \in N(v)} \alpha_{vu}^k W^k h_u^{k-1}\right),$$

where α_{vu}^k is learned by an attention mechanism.
A common attention score is

$$e_{vu} = \text{LeakyReLU}\left(a^\top [Wh_v \| Wh_u]\right), \quad = \alpha(W_k h_u^{k-1}, W_k h_v^{k-1})$$

followed by normalization:

$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{s \in N(v)} \exp(e_{vs})}.$$

Here a is a trainable attention vector, W is a trainable feature transformation matrix, $\|$ denotes vector concatenation, e_{vu} is the unnormalized attention score, and α_{vu} is the normalized importance of neighbor u to node v.
With multi-head attention, the layer can be written as

$$h_v^k = \Big\|_{r=1}^{R} \sigma\left(\sum_{u \in N(v)} \alpha_{vu}^{k,r} W^{k,r} h_u^{k-1}\right), \quad h_v^k = \sigma\left(\sum_{u \in N(v)} \alpha_{vu} W_k h_u^{k-1}\right)$$

where R is the number of attention heads, r indexes the head, and $\alpha_{vu}^{k,r}$ is the attention weight from node u to node v in head r at layer k. Multi-head attention stabilizes learning and improves expressiveness.

2.9 Comparison of Graph Neural Models

Method	Core Mechanism	Problem Solved	Limitation
Naive DNN on A, X	Concatenate adjacency and features	Directly uses graph input	Fixed graph size; no permutation invariance; parameters depend on N
Early graph convolution	Canonical ordering of local subgraphs	Tries to transfer CNN filters to graphs	Canonical ordering is hard to define
Basic GCN	Mean neighbor aggregation with shared weights	Permutation-invariant; stackable; inductive to new nodes	All neighbors are equally weighted
GraphSAGE	Generalized aggregation and neighbor sampling	Better inductive learning and scalability	Aggregator design still matters
GAT	Attention-based aggregation	Learns neighbor importance	More expensive; attention is not always interpretable

3 Part II: From Tool Use to Skills

3.1 Core Problem: Why Does an LLM Need Tool Use? [Very Important]

A language model is fundamentally a pure function:

$$\text{completion} = f_\theta(\text{prompt}).$$

It takes text as input and produces text as output. It has no persistent state, no side effects, and no direct access to files, shell commands, APIs, code editing, or web search.
The key design principle of tool use is

model decides, host executes.

The model reads the task, selects a tool, and emits a structured call. The host executes the actual operation and returns the result. This separates decision-making from side effects.

3.2 The Four-Step Tool-Use Protocol [Very Important]

A complete tool call follows a four-step handshake.

3.2.1 Step 1: Declare Tools

The host sends the user message together with a list of available tool schemas:

$$\text{request} = (\text{messages}, \text{tools}[]).$$

Each tool declaration contains

$$\text{name}, \quad \text{description}, \quad \text{input\_schema}.$$

The model sees the shape of the tool, not its implementation. For example, it may know that a tool named `get_weather` requires an input field `city:string`, but it does not know how the weather is actually fetched.

3.2.2 Step 2: The Model Emits `tool_use`

Instead of directly producing a final answer, the model emits a structured request:

$$\text{tool\_use} = (\text{id}, \text{name}, \text{input}).$$

Here id correlates the request and result, name is the selected tool, and input is a JSON object conforming to the schema.

3.2.3 Step 3: The Host Executes

The host parses the tool call, performs the real operation, and obtains

$$\text{output} = \text{dispatch}(\text{name}, \text{input}).$$

It then wraps the output as

$$\text{tool\_result} = (\text{tool\_use\_id}, \text{content}),$$

where tool_use_id must match the previous tool-use id.

3.2.4 Step 4: The Model Closes the Turn

After receiving tool_result, the model generates a natural-language reply:

$$\text{tool\_result} \rightarrow \text{final answer}.$$

If more external information is needed, the model may emit another tool call. Thus, one user turn does not necessarily correspond to one tool call.

3.3 The Agentic Loop [Very Important]

The agentic loop is

$$\begin{aligned} &\text{context+tools} \rightarrow \text{model decides} \rightarrow \text{tool\_use} \rightarrow \text{host executes} \\ &\rightarrow \text{tool\_result} \rightarrow \text{model decides again} \rightarrow \cdots \rightarrow \text{end\_turn}. \end{aligned}$$

The loop may stop because the model outputs `end_turn`, the host interrupts execution, a maximum turn limit is reached, or permission is rejected.

This is the key difference between a chatbot and an agent. A chatbot produces an answer in one generation. An agent forms a closed-loop control system between model reasoning and external execution.

3.4 Claude Code as an Agentic Harness [Important]

Claude Code can be understood as

$$\text{Claude Code} = \text{Model} + \text{Tools} + \text{Permissions} + \text{UI} + \text{Context Management} + \text{Hooks} + \text{Sub-agents} + \text{MCP Client}.$$

Its client-side tools include Bash, Read, Write, Edit, Grep, Glob, WebFetch, and Task. The model does not directly execute Bash. It emits a Bash tool call; the host checks permissions and executes it.

3.5 MCP: External Tools Under the Same Protocol [Important]

MCP allows external systems to expose their capabilities as tools under the same protocol. For example, Jira MCP, Postgres MCP, and GitHub MCP can all be merged into the same tools[] array. From the model's perspective, built-in tools and MCP tools are both structured tools. The difference lies mainly in where and how execution happens:

$$\text{built-in tool} \approx \text{MCP tool}$$

at the protocol level.

3.6 Why Tools Alone Do Not Scale [Very Important]

Tools create two scalability problems.

First, **context bloat**: every tool name, description, and input schema must be placed into the model context. With many tools or MCP servers, the tool definitions alone may consume a large portion of the context window.

Second, **operation manuals**: many tasks do not need a new executable capability, but need domain-specific instructions. For example, writing a Word document may not require a new tool, but it does require a workflow or style guide. If every manual is placed in the system prompt, every unrelated task pays the context cost.

The desired mechanism is

$$\text{instructions should appear only when needed.}$$

3.7 Skills: A Wrapper, Not a New Mechanism [Very Important]

The key idea is

$$\text{A skill is just a wrapper.}$$

More concretely,

$$\text{Skill(command:string)} \rightarrow \text{return SKILL.md body.}$$

A skill is not a new reasoning mechanism and not a new external API. It is a generic tool call that asks the host to read a corresponding `SKILL.md` file from disk and append its markdown content to the context as a tool result.

Therefore, the essence of skills is

$$\text{on-demand context loading.}$$

Skills solve the problem of when domain instructions should enter the context, not the problem of how to execute side effects.

3.8 Skill Loading: Catalog Always Loaded, Body On Demand [Very Important]

A skill has two levels of content.

The first level is the catalog: skill name and description. This is preloaded in the system prompt as prose, not as a separate tool schema.

The second level is the full body of `SKILL.md`. This is loaded only when the model invokes the skill.

Formally,

$$\text{before invocation:} \quad \text{context} \leftarrow \{\text{name}, \text{description}\},$$

$$\text{on invocation:} \quad \text{context} \leftarrow \text{context} + \text{SKILL.md body.}$$

Even if there are many skills, they share one generic `Skill` tool schema:

$$\text{Skill(skill, args).}$$

Each skill does not need its own JSON schema. The trigger logic is mostly expressed through natural-language descriptions.

3.9 Built-in Tool, MCP Tool, Skill, and Sub-agent

Primitive	Core Idea	Use When	Avoid When
Tool	New executable capability	File I/O, APIs, shell, code execution, computation	Long domain manuals
MCP	Tools from external systems	GitHub, Slack, databases, Jira, internal services	Pure documentation or style guides
Skill	On-demand manual	Domain knowledge, workflows, style guides	Directly sending emails or calling APIs
Sub-agent	Fresh context, isolation, parallelism	Long tasks, multi-role systems, information isolation	Simple single-step tool calls

The design rule is: an action such as sending an email should be a tool or MCP capability, not a skill. A domain style guide should be a skill, not a permanent system prompt. A very long workflow should not be one giant `SKILL.md`; it should be split into sub-agents or chained skills.

4 Unified View: Shared Computation and On-Demand Expansion

Although the two lectures discuss different systems, they share a similar design principle.

GCNs solve the problem that graphs are large, unordered, and variable-sized. Instead of assigning independent parameters to each node, they learn a shared local aggregation function:

$$z_v = f_\theta(v, X, A).$$

Skills solve the problem that agents have too many tools, manuals, and workflows to keep permanently in context. Instead of placing all instructions in the prompt, they load only the relevant manual:

$$\text{needed skill} \Rightarrow \text{load only the relevant SKILL.md.}$$

The common principle is:

> Do not hard-code everything into a fixed input. Use shared, compositional, and on-demand structures.

In GCNs, the computation graph expands according to the node neighborhood. In agents, the context expands according to the task.

5 Final Exam Priority List

Highest Priority

You must master the following points.

- Why a naive DNN over A and X fails: graph size varies, node ordering is arbitrary, and the model lacks permutation invariance.
- The basic GCN formula:

$$h_v^k = \sigma\left(W_k \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + B_k h_v^{k-1}\right).$$

- The meaning of every symbol in the GCN formula.
- Why K GCN layers correspond to K-hop information propagation.
- Why shared parameters enable inductive generalization.
- Why GraphSAGE generalizes mean aggregation.
- Why GAT introduces learnable attention weights α_{vu}.
- The four steps of tool use:

$$\text{declare} \rightarrow \text{tool\_use} \rightarrow \text{execute} \rightarrow \text{tool\_result/reply.}$$

- Why one agent turn can include multiple tool calls.
- Why a skill is not a new executable capability, but an on-demand manual-loading mechanism.
- The differences among Tool, MCP, Skill, and Sub-agent.

Medium Priority

- Unsupervised graph embedding losses based on random walks.
- Supervised node classification loss.
- Claude Code as an agentic harness.
- Catalog/body separation in the skill mechanism.

Lower Priority but Useful for Bonus Understanding

- Large-scale dynamic graph examples such as Pinterest/Pixie.
- Complex agent case studies such as PokerBot, where skills, tools, and sub-agents are composed.

Week 12 Study Notes: Information Diffusion and Recommendation in Social Media

1 Global Roadmap: From Diffusion to Recommendation

[Importance: ⋆⋆⋆⋆⋆]

Although the two slide decks appear to cover different topics, they share the same underlying question: **how individual-level states or preferences evolve into system-level patterns**. Information diffusion asks how a disease, virus, rumor, tweet, or behavior spreads through a population. Recommendation asks how a system predicts future user preferences from historical interactions.

The central conceptual tree is:

- Social-media information flow
- Information diffusion / epidemic modeling: implicit network, unknown infection paths; SI: infection is irreversible; SIR: infected users recover or are removed; SIS: recovered users become susceptible again; SIRS: immunity decays after recovery.
- Recommendation systems: search vs. recommendation; content-based recommendation; collaborative filtering; user-based CF; item-based CF; model-based CF / SVD; evaluation by predictive, classification, and ranking accuracy.

2 Part I: Information Diffusion via Epidemic Models

2.1 Core Objective and Modeling View

[Importance: ⋆⋆⋆⋆⋆]

The motivating example is the Melissa computer worm. It started in March 1999, infected Microsoft Outlook users, and spread when a user opened an infected Word document. Once opened, the virus sent itself to the first 50 users in the Outlook address book. It was first detected on Friday, March 26, and by Monday, March 29, more than 100,000 computers had been infected. The point is not the specific virus, but the abstraction: disease, malware, tweets, rumors, and social behaviors can all be treated as spreading objects.

An epidemic process consists of three components:

$$\text{Epidemic process} = \text{pathogen} + \text{hosts} + \text{spreading mechanism}.$$

Epidemiological term	Social-media analogue
Pathogen	Disease, computer virus, tweet, meme, rumor, or behavior being spread.
Hosts	Humans, accounts, users, devices, animals, plants, etc.
Spreading mechanism	Breathing, physical contact, retweeting, sharing, email forwarding, purchasing/selling links.

隐式 不知谁传谁

The key modeling distinction is that epidemic models usually assume an **implicit network** and **unknown connections between individuals**. Therefore, they are appropriate when we care about global patterns, such as the trend and ratio of infected people, rather than identifying exactly who infected whom. This differs from cascade models, which are more concerned with explicit propagation chains.

2.2 Contact Network vs. Fully Mixed Model

[Importance: ⋆⋆⋆⋆⋆]

There are two broad ways to analyze epidemics.

Method	Assumption	Focus	Strength	Limitation
Contact network	We know which hosts contact which other hosts.	Who contacts whom; who infects whom.	Fine-grained; can model network topology.	Requires contact-network data.
Fully mixed model	We do not use explicit network information.	Aggregate rates of infection and recovery	Simple; suitable for macro-level trends.	Ignores heterogeneity and structural effects. 异质性

The slides focus on fully mixed models, assuming that no contact-network information is available and that the exact infection mechanism is unknown.

只建模 S(t), I(t), R(t)等

2.3 SI Model: Susceptible to Infected

[Importance: ⋆⋆⋆⋆⋆]

2.3.1 Definition

The SI model has two states:

$$S \longrightarrow I.$$

Here, S denotes susceptible individuals: those who are not infected but can potentially become infected. I denotes infected individuals: those who have been infected and can infect susceptible individuals. Once infected, individuals never recover. Hence SI is suitable for irreversible adoption processes, such as permanent awareness or permanent adoption of some information.

2.3.2 Notation

Symbol	Meaning
N	Total population size.
$S(t)$	Number of susceptible individuals at time t.
$I(t)$	Number of infected individuals at time t.
$s(t) = S(t)/N$	Fraction of susceptible individuals.
$i(t) = I(t)/N$	Fraction of infected individuals.
β	Contact probability or transmission intensity. If $\beta = 1$, everyone contacts everyone else; if $\beta = 0$, no one contacts anyone else.
I_0	Initial number of infected individuals, i.e., $I(0)$.

The population is conserved:

$$N = S(t) + I(t).$$

2.3.3 SI Dynamics

At each time step, one infected individual meets βN people on average. Among these, the susceptible fraction is S/N, so one infected individual infects approximately

$$\beta N \cdot \frac{S}{N} = \beta S$$

susceptible individuals. Since there are I infected individuals, the total number of new infections is

$$\beta IS.$$

Therefore,

$$\frac{dS}{dt} = -\beta IS, \tag{1}$$

$$\frac{dI}{dt} = \beta IS. \tag{2}$$

The first equation says susceptible individuals are consumed by infection; the second says the infected population grows by the same amount.

2.3.4 Derivation of Logistic Growth

Using $S = N - I$, we obtain

$$\frac{dI}{dt} = \beta I(N - I) \tag{3}$$

$$= \beta NI - \beta I^2. \tag{4}$$

This is the logistic growth equation. Its solution is

$$I(t) = \frac{NI_0 e^{\beta Nt}}{N + I_0\left(e^{\beta Nt} - 1\right)}. \tag{5}$$

The solution is S-shaped: early growth is slow because I is small; middle-stage growth is rapid because both S and I are substantial; late-stage growth slows because susceptible individuals are exhausted.

2.4 SIR Model: Adding Recovery or Removal

[Importance: ⋆⋆⋆⋆⋆]

2.4.1 Why SI Fails

SI assumes that infected individuals never recover. This is unrealistic for many systems: people recover from diseases, infected devices may be cleaned, users may stop forwarding a piece of information, and contagious status is often temporary. SIR fixes this by adding a recovered or removed state.

2.4.2 Definition

The state transition is

$$S \longrightarrow I \longrightarrow R.$$

The total population is conserved:

$$S(t) + I(t) + R(t) = N.$$

Recovered or removed individuals cannot become infected again and do not transmit the disease.

2.4.3 SIR Equations

The model is

$$\frac{dS}{dt} = -\beta IS, \tag{6}$$

$$\frac{dI}{dt} = \beta IS - \gamma I, \tag{7}$$

$$\frac{dR}{dt} = \gamma I. \tag{8}$$

Here, β is the infection or contact-transmission rate, and γ is the recovery probability or recovery rate. The infected population changes according to

$$\frac{dI}{dt} = \text{new infections} - \text{recoveries} = \beta IS - \gamma I.$$

Thus $I(t)$ increases when $\beta S > \gamma$ and decreases when $\beta S < \gamma$.

2.4.4 Key Derivation and Absence of a Closed Form

Divide dS/dt by dR/dt:

$$\frac{dS}{dR} = \frac{dS/dt}{dR/dt} \tag{9}$$

$$= \frac{-\beta IS}{\gamma I} \tag{10}$$

$$= -\frac{\beta}{\gamma} S. \tag{11}$$

Separate variables:

$$\frac{dS}{S} = -\frac{\beta}{\gamma} dR.$$ 两边积分

Integrate:

$$\log S = -\frac{\beta}{\gamma} R + C.$$

If $R_0 = 0$ and $S(0) = S_0$, then $C = \log S_0$, giving

$$\log \frac{S_0}{S} = \frac{\beta}{\gamma} R, \tag{12}$$

or equivalently,

$$S = S_0 e^{-\frac{\beta}{\gamma} R}. \tag{13}$$

Since $I = N - S - R$, we have

$$\frac{dR}{dt} = \gamma I \tag{14}$$

$$= \gamma(N - S - R) \tag{15}$$

$$= \gamma \left(N - S_0 e^{-\frac{\beta}{\gamma} R} - R \right). \tag{16}$$

Thus

$$dt = \frac{1}{\gamma} \frac{dR}{N - S_0 e^{-\frac{\beta}{\gamma} R} - R}, \tag{17}$$

and

$$t = \frac{1}{\gamma} \int_0^R \frac{dx}{N - S_0 e^{-\frac{\beta}{\gamma} x} - x}. \tag{18}$$

This integral has no general closed-form solution, so numerical approximation is required.

2.5 SIS Model: Recovery Back to Susceptibility

[Importance: ⋆⋆⋆⋆]

2.5.1 Why SIR Fails

SIR assumes permanent immunity after recovery. Many processes violate this assumption: people can catch similar infections again, malware may reinfect a system, and users may repeatedly become susceptible to the same type of information. SIS therefore assumes

$$S \longrightarrow I \longrightarrow S.$$

恢复者 立马易感

2.5.2 SIS Equations

The equations are

$$\frac{dS}{dt} = \gamma I - \beta IS, \tag{19}$$

$$\frac{dI}{dt} = \beta IS - \gamma I. \tag{20}$$

The term βIS moves individuals from S to I, while γI moves infected individuals back to S. Since $S = N - I$,

$$\frac{dI}{dt} = \beta I(N - I) - \gamma I \tag{21}$$

$$= I(\beta N - \gamma) - \beta I^2. \tag{22}$$

2.5.3 Threshold Behavior

If

$$\beta N \leq \gamma,$$

then $I(\beta N - \gamma) \leq 0$ and $-\beta I^2 \leq 0$, so $dI/dt < 0$ whenever $I > 0$. Thus $I(t)$ decreases exponentially toward zero. Intuitively, recovery dominates transmission.

If

$$\beta N > \gamma,$$

then the equation has a logistic-growth form and admits a nonzero endemic equilibrium. Setting $dI/dt = 0$ gives

$$I(\beta N - \gamma) - \beta I^2 = 0, \tag{23}$$

非零稳定态。

$$I\left[(\beta N - \gamma) - \beta I\right] = 0. \tag{24}$$

Therefore,

$$I^* = 0 \tag{25}$$

or

$$I^* = N - \frac{\gamma}{\beta}. \tag{26}$$

The nonzero equilibrium exists only when $\beta N > \gamma$.

2.6 SIRS Model: Immunity Decay

[Importance: ⋆⋆⋆⋆]

SIRS combines SIR and SIS by inserting a temporary recovered state:

$$S \longrightarrow I \longrightarrow R \longrightarrow S.$$

Recovered individuals lose immunity after some period and become susceptible again. The equations are

$$\frac{dS}{dt} = \lambda R - \beta IS, \tag{27}$$

$$\frac{dI}{dt} = \beta IS - \gamma I, \tag{28}$$

$$\frac{dR}{dt} = \gamma I - \lambda R. \tag{29}$$

变为易感者。

Here, λ is the immunity-loss rate. The term λR moves recovered individuals back to susceptibility, γI moves infected individuals to recovery, and βIS moves susceptible individuals to infection. Like SIR, SIRS generally has no closed-form solution and is usually studied by numerical integration.

Model	Transition	Recovery?	Repeat infection?	Closed form?	Typical use
SI	$S \to I$	No	No	Yes	Irreversible adoption or permanent infection.
SIR	$S \to I \to R$	Yes	No	Generally no	One-time infection followed by immunity or removal.
SIS	$S \to I \to S$	Yes	Yes	Partly analyzable; logistic-like	Recurrent infection without long-term immunity.
SIRS	$S \to I \to R \to S$	Yes	Yes, after immunity decay	Generally no	Temporary immunity followed by renewed susceptibility.

2.7 Comparison of Epidemic Models

[Importance: ⋆⋆⋆⋆⋆]

The methodological evolution is:

$$\text{SI} \xrightarrow{\text{add recovery/removal}} \text{SIR} \xrightarrow{\text{allow reinfection}} \text{SIS} \xrightarrow{\text{add temporary immunity}} \text{SIRS}.$$

2.8 Epidemic Intervention: Mad Cow Disease

[Importance: ⋆⋆⋆]

The mad cow disease example illustrates intervention through weak ties. In January 2001, the first case was observed in the UK; by February 2001, 43 farms were infected; by September 2001, around 9000 farms were infected. Weak ties included animal trade and indirect contact such as contaminated soil carried by tourists. The interventions were to ban movement, making contagion harder, and to kill infected or exposed animals, effectively removing weak ties or infectious nodes from the system.

3 Part II: Recommendation in Social Media

3.1 Core Objective of Recommender Systems

[Importance: ⋆⋆⋆⋆⋆]

The fundamental assumption is that user preferences tend to remain stable and change smoothly over time. Therefore, a system can use historical behavior, such as past ratings, purchases, clicks, or the behavior of similar users, to predict future interests.

Formally, a recommender system learns

$$f : U \times I \to R, \tag{30}$$

where U is the set of users, I is the set of items, R is the rating or preference space, and $f(u, i)$ is the predicted preference of user u for item i.

Recommendation differs from search. Search is query-driven and often returns the same results for the same query, such as "best 2014 movie to watch," regardless of whether the user is a child or an adult. Recommendation is user-driven and personalized.

3.2 Challenges of Recommender Systems

[Importance: ⋆⋆⋆⋆]

Challenge	Meaning	Why it is difficult
Cold start	A new user or item has little or no history.	The system cannot infer preferences from past behavior.
Data sparsity	Historical information is insufficient system-wide.	Similarity estimates and model learning become unstable.
Attacks	Malicious behavior manipulates recommendations.	Fake users can conduct push attacks; nuke attacks can disrupt the system.
Privacy	Recommendation may use private information.	User data can be exposed or misused.
Explanation	Recommendations may lack justification.	Users may not trust unexplained recommendations.

Cold start is an individual-level issue; data sparsity is a system-level issue.

3.3 Content-Based Recommendation

[**Importance:** ⋆⋆⋆⋆⋆]

3.3.1 Assumption and Algorithm

The assumption is that a user should be recommended items whose descriptions match the user's interests. The more similar an item description is to the user's profile, the more likely the user will find it interesting.

The algorithm is:

1. Describe the items to be recommended.
2. Create a user profile describing the types of items the user likes.
3. Compare each item with the user profile.
4. Recommend the most similar items.

The user profile is often created and updated automatically according to feedback on presented items.

3.3.2 Vector Representation and Cosine Similarity

Using k keywords, represent item j as

$$I_j = (i_{j,1}, i_{j,2}, \ldots, i_{j,k}), \tag{31}$$

and user i as

$$U_i = (u_{i,1}, u_{i,2}, \ldots, u_{i,k}). \tag{32}$$

Here, $i_{j,l}$ is the weight of keyword l in item j, and $u_{i,l}$ is the weight of keyword l in user i's profile. These weights may be computed by TF–IDF.

The cosine similarity is

$$sim(U_i, I_j) = \cos(U_i, I_j) = \frac{\sum_{l=1}^{k} u_{i,l} i_{j,l}}{\sqrt{\sum_{l=1}^{k} u_{i,l}^2}\sqrt{\sum_{l=1}^{k} i_{j,l}^2}}. \tag{33}$$

The numerator measures overlap between user interest and item content; the denominator normalizes vector length to avoid favoring long descriptions or high-frequency keywords.

3.3.3 Limitations

Content-based recommendation is interpretable and does not require other users' ratings, but it has several limitations: it requires item descriptions, it may over-specialize by recommending only items similar to past interests, it cannot exploit collective user behavior, it still faces cold start for new users, and its quality depends heavily on feature representation.

3.4 Collaborative Filtering

[**Importance:** ⋆⋆⋆⋆⋆]

3.4.1 Core Idea

Collaborative filtering selects information or patterns using collaboration among multiple agents, viewpoints, or data sources. Its advantage is that it does not require additional information about users or item content. User ratings or purchase history are sufficient.

Ratings may be explicit, such as direct 1–5 star ratings, or implicit, such as playlists, listening history, purchases, clicks, or time spent on a webpage.

3.4.2 Rating Matrix

Let

$$R \in \mathbb{R}^{m \times n} \tag{34}$$

be the user–item matrix, where m is the number of users, n is the number of items, and $r_{u,i}$ is user u's rating of item i. Most entries are missing. Recommendation is therefore a missing-rating prediction problem.

3.4.3 Memory-Based vs. Model-Based CF

Type	Idea	Strength	Limitation
Memory-based CF	Directly use the stored rating matrix and similarity between users or items.	Simple and interpretable.	Expensive and unstable under sparsity.
Model-based CF	Assume an underlying model governs ratings and learn it.	Better generalization and denoising.	Requires training and may be less interpretable.

3.5 Memory-Based Collaborative Filtering

[**Importance:** ⋆⋆⋆⋆⋆]

There are two major memory-based methods. User-based CF assumes that users with similar previous ratings are likely to rate future items similarly. Item-based CF assumes that items receiving similar ratings from users are likely to receive similar ratings from future users.

3.5.1 General Algorithm

1. Weight all users or items by similarity to the current user or item.
2. Select a subset of users or items as neighbors.
3. Predict the rating of a user for a specific item using neighbors' ratings for the same or similar items.
4. Recommend items with the highest predicted ranks.

3.5.2 User-Based Collaborative Filtering

The system finds users most similar to the current user and uses their preferences for recommendation. The prediction formula is

$$r_{u,i} = \bar{r}_u + \frac{\sum_{v \in N(u)} sim(u,v)(r_{v,i} - \bar{r}_v)}{\sum_{v \in N(u)} sim(u,v)} \tag{35}$$

Here, $r_{u,i}$ is the predicted rating of user u for item i, $\bar{r}_u$ is user u's mean rating, $N(u)$ is the neighbor set of user u, $sim(u,v)$ is the similarity between users u and v, $r_{v,i}$ is user v's observed rating for item i, and $\bar{r}_v$ is user v's mean rating.

The term $r_{v,i} - \bar{r}_v$ captures whether user v likes item i more or less than their own average. Adding $\bar{r}_u$ maps this relative preference back to user u's rating scale. This mean-centering is important because different users have different rating biases.

User-based CF is less popular than item-based CF in large systems because user populations are large and dynamic. Even small changes in user data may reset the entire group of similar users.

3.5.3 Similarity Measures

Cosine similarity is

$$sim(U_u, U_v) = \cos(U_u, U_v) = \frac{U_u \cdot U_v}{\|U_u\| \, \|U_v\|} = \frac{\sum_i r_{u,i} r_{v,i}}{\sqrt{\sum_i r_{u,i}^2} \sqrt{\sum_i r_{v,i}^2}}. \tag{36}$$

It measures the directional similarity between two rating vectors.

Pearson correlation coefficient is

$$sim(U_u, U_v) = \frac{\sum_i (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_i (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_i (r_{v,i} - \bar{r}_v)^2}}. \tag{37}$$

Pearson similarity removes user-specific rating bias by subtracting each user's mean rating.

3.5.4 Item-Based Collaborative Filtering

Item-based CF first computes similarity between items, then predicts a target rating using the user's ratings on similar items. A standard mean-centered form is

$$r_{u,i} = \bar{r}_i + \frac{\sum_{j \in N(i)} sim(i,j)(r_{u,j} - \bar{r}_j)}{\sum_{j \in N(i)} sim(i,j)}. \tag{38}$$

Here, $\bar{r}_i$ is item i's mean rating, $N(i)$ is the set of items most similar to item i, $sim(i,j)$ is item–item similarity, $r_{u,j}$ is user u's rating of similar item j, and $\bar{r}_j$ is item j's mean rating.

Item-based CF is often more stable than user-based CF because item similarities change more slowly than user neighborhoods.

3.6 Model-Based CF and SVD

[**Importance:** ⋆⋆⋆⋆⋆]

3.6.1 Motivation

Memory-based CF directly relies on observed similarities. It becomes unreliable when the rating matrix is sparse, expensive at large scale, sensitive to noisy ratings, and unable to explicitly model latent structure. Model-based CF assumes that an underlying model governs how users rate items. This model is learned and then used to predict missing ratings.

3.6.2 Singular Value Decomposition

Given

$$X \in \mathbb{R}^{m \times n}, \qquad m \geq n, \tag{39}$$

SVD factorizes X as

$$X = U\Sigma V^T. \tag{40}$$

Here, $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal matrices, and $\Sigma \in \mathbb{R}^{m\times n}$ is diagonal. The product is exactly the original matrix, so full SVD loses no information.

3.6.3 Low-Rank Matrix Approximation

A low-rank approximation of X is another matrix

$$C \in \mathbb{R}^{m\times n} \tag{41}$$

with fixed rank

$$Rank(C) = k, \qquad k \ll \min(m, n). \tag{42}$$

The best low-rank approximation minimizes

$$\min_C \|X - C\|_F, \tag{43}$$

where the Frobenius norm is

$$\|X - C\|_F = \sqrt{\sum_{u=1}^{m}\sum_{i=1}^{n}(X_{u,i} - C_{u,i})^2}. \tag{44}$$

SVD computes the best rank-k approximation

$$X_k = U_k\Sigma_k V_k^T. \tag{45}$$

The intuition is that the rating matrix is not random; it has an underlying low-dimensional structure. Keeping the top k singular values captures dominant latent factors while removing noise.

3.7 Social Recommendation

[Importance: ⋆⋆⋆⋆]

Classical recommendation allows any similar user or item to contribute to prediction. Social recommendation constrains the contributing users using social context. Let $F(i)$ denote the friends of user i, and let $sim(i, j)$ denote similarity between users i and j. The system can restrict recommendation contributors to $S(i)$, the set of the k most similar friends of individual i.

The intuition is that not all similar users are equally trustworthy; similar friends may be more informative than arbitrary similar strangers. This can help control sparsity and make recommendations more socially grounded.

4 Evaluation of Recommender Systems

[Importance: ⋆⋆⋆⋆⋆]

Evaluating recommender systems is difficult because different algorithms perform differently across datasets, and application goals vary. Some datasets have many more users than items, or many more items than users. Rating density, rating scale, and domain properties also matter. Early evaluation focused on predicting withheld ratings, but user satisfaction, response time, purchases, returns, and business outcomes are also important.

Application-level outcomes include add-on sales, click-through rates, number of purchased products, return behavior, and user satisfaction. Offline research metrics attempt to anticipate these outcomes.

4.1 Predictive Accuracy

[Importance: ⋆⋆⋆⋆⋆]

Predictive accuracy asks how close predicted ratings are to true ratings.

Mean Absolute Error is

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|p_i - r_i|, \tag{46}$$

where p_i is the predicted rating, r_i is the true rating, and n is the number of test examples.

Normalized MAE is

$$NMAE = \frac{MAE}{r_{max} - r_{min}}, \tag{47}$$

where r_{max} and r_{min} are the maximum and minimum possible ratings.

Root Mean Square Error is

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_i - r_i)^2}. \tag{48}$$

RMSE penalizes large errors more strongly because deviations are squared.

4.2 Classification Accuracy

[**Importance:** ⋆⋆⋆⋆⋆]

When the task is to find good items rather than predict exact ratings, precision and recall are appropriate.

Precision measures exactness:

$$Precision = \frac{|\text{relevant items retrieved}|}{|\text{all items retrieved}|}. \tag{49}$$

Recall measures completeness:

$$Recall = \frac{|\text{relevant items retrieved}|}{|\text{all relevant items}|}. \tag{50}$$

Precision asks: among recommended items, how many are truly relevant? Recall asks: among all relevant items, how many did the system retrieve?

4.3 Ranking Accuracy

[**Importance:** ⋆⋆⋆⋆⋆]

Recommendation usually outputs ranked lists, so ranking accuracy is essential.

Spearman's rank correlation is

$$\rho = 1 - \frac{6\sum_{i=1}^{n}(x_i - y_i)^2}{n^3 - n}, \tag{51}$$

where x_i is the predicted rank of item i, y_i is its true rank, and n is the number of ranked items. Perfect agreement gives $\rho = 1$.

Kendall's τ is

$$\tau = \frac{c - d}{\binom{n}{2}}, \tag{52}$$

where c is the number of concordant item pairs, d is the number of discordant item pairs, and $\binom{n}{2}$ is the total number of pairs. A pair is concordant if the predicted and true rankings agree on its relative order; otherwise, it is discordant.

5 Methodological Evolution Summary

[**Importance:** ⋆⋆⋆⋆⋆]

5.1 Epidemic Models

Step	Model	Failure of previous model	Fix
1	SI	No recovery; infection is permanent.	Baseline irreversible diffusion.
2	SIR	SI cannot model recovery or removal.	Add R state.
3	SIS	SIR assumes permanent immunity.	Return recovered individuals to S.
4	SIRS	SIS lacks an explicit immunity period.	Add $R \to S$ immunity decay.

5.2 Recommendation Algorithms

Search $\xrightarrow{\text{needs personalization}}$ Recommendation $\xrightarrow{\text{content features}}$ Content-based

$\xrightarrow{\text{needs collective behavior}}$ Collaborative filtering $\xrightarrow{\text{sparsity and noise}}$ Model-based CF / SVD

$\xrightarrow{\text{social constraints}}$ Social recommendation.

Method	Core assumption	Why it helps	Main weakness
Search	Results match a query.	Simple and direct.	Not personalized.
Content-based	Users like items similar to their profiles.	Interpretable; uses item descriptions.	Over-specialization; needs features.
User-based CF	Similar users like similar items.	Uses collective behavior.	Unstable for many dynamic users.
Item-based CF	Similar items receive similar ratings.	More stable item similarities.	Still suffers from sparsity.
Model-based CF / SVD	Ratings are generated by latent factors.	Denoises and generalizes.	Requires model learning.
Social recommendation	Friends or similar friends are more trustworthy.	Uses social context and controls sparsity.	Depends on quality of social links.

Topic	Exam requirement
SI equation and logistic solution	Derive $\frac{dI}{dt} = \beta I(N - I)$ from $S + I = N$ and explain the S-shaped curve.
SIR equations	Explain βIS and γI and the condition for $I(t)$ to increase.
SIR dS/dR derivation	Derive $\log(S_0/S) = \frac{\beta}{\gamma}R$ and explain why closed form is unavailable.
SIS threshold	Explain $\beta N \leq \gamma$ versus $\beta N > \gamma$ and the nonzero equilibrium $I^* = N - \gamma/\beta$.
Content-based cosine similarity	Write the formula and explain TF–IDF user/item vectors.
User-based CF prediction	Explain mean-centering and neighbor-weighted prediction.
Cosine vs. Pearson	Compare whether rating bias is removed.
SVD low-rank approximation	Explain $X_k = U_k \Sigma_k V_k^T$ and why low-rank structure denoises ratings.
Evaluation metrics	Distinguish MAE, RMSE, Precision, Recall, Spearman ρ, and Kendall τ.

6 Exam-Focused Priority List

Highest Priority: Must Derive or Explain

Medium Priority

Topic	Requirement
Epidemic vs. cascade	Understand implicit network versus explicit propagation chain.
Contact network vs. fully mixed	Understand macro-level modeling assumptions.
Item-based CF	Understand why it is more stable than user-based CF.
Social recommendation	Understand how friend constraints can reduce sparsity and improve contextual relevance.
Cold start vs. data sparsity	Distinguish individual-level and system-level lack of information.

Lower Priority but Possible Concept Questions

Topic	Requirement
Melissa worm	Use as an analogy for epidemic spreading.
Mad cow disease intervention	Understand banning movement and removing infected nodes/weak ties.
Push attack and nuke attack	Know security threats to recommender systems.
Privacy and explanation	Know non-accuracy challenges in recommendation.

Geometric Centrality 都是node对其他V上所有

① Degree $C_d(v_i) = d_i$ $n-1$, $\max_j d_j$, $\sum_j d_j = 2|E|$

② closeness $C_c(v_i) = \frac{1}{\frac{1}{n-1}\sum_v l_{ij}}$

③ Harmonic $C_{har}(v_i) = \sum_{v_j \in V} \frac{1}{l_{ij}}$

Spectral ① Eigen $C_e(v_i) = \frac{1}{\lambda}\sum_{j=1}^{n} A_{ji} C_e(v_j)$ $C_e = \frac{1}{\lambda} A^T C_e$

初始化为1, $C_e(v_i) \leftarrow \sum_{j=1}^{n} A_{ji} C_e(v_j)$ $C_e(v_i) \leftarrow \frac{C_e(v_i)}{\sqrt{\sum_{j=1}^{n} (C_e(v_j))^2}}$

$O(n^3)$ 算 eigen

② Katz $C_{katz} = \alpha \sum_{j=1}^{n} A_{ji} (C_{katz}(v_j) + \beta$ $C = \alpha A^T C + \beta 1$ $C = \beta (I - \alpha A^T)^{-1} 1$

初始化为1 $C_{katz}(v_i) \leftarrow \sum_{j=1}^{n} A_{ji} C_{katz}(v_j) + \frac{\beta}{\alpha}$ $C_{katz}(v_i) \leftarrow \frac{C_{katz}(v_i)}{\sum_j (C_{katz}(v_j))^2}$

③ PageRank $C_p(v_i) = \alpha \sum_{j=1}^{n} \frac{A_{ji} C_p(v_j)}{d_j^{out}} + \beta$ $\beta = 1-\alpha$

norm $C_p(v_i) \leftarrow \frac{C_p(v_i)}{\sum_j C_p(v_j)}$ 非1z

Global CC: $\frac{\#\text{有三边的二边 paths}}{\#\text{长2 path}} = \frac{3 \times \#\text{三角形数}}{3 \times \#\text{三角形数} + \#\text{剩余 triplet}}$

Local CC $C(v_i) = \frac{\#\text{邻居相连对数}}{\frac{d_i(d_i-1)}{2}}$

Average CC $C(G) = \frac{1}{|V|}\sum_{v \in V} C(v)$ $a \to b$ 算1, $b \to a$ 算1.

Reciprocity $R = \frac{2 \times \#\text{Reciprocal node pairs}}{\#\text{ of edges}}$ ↙ $a \leftrightarrow b$

Jaccard $\frac{|N(v_i) \cap N(v_j)|}{|N(v_i) \cup N(v_j)|}$ cos $\frac{|N(v_i) \cap N(v_j)|}{\sqrt{|N(v_i)||N(v_j)|}}$

Power law $f(k) = a k^{-b}$

	现实	RG	Small World	Prefrential
度分布	P-L	Bino	XP-L	P-L
CC	High	low(p)	High $\frac{3(d-2)}{4(d-1)}$	low
Avg shortest	Small	Small $\frac{\ln\|V\|}{\ln d}$	Small $\frac{n}{2d}$ 直径 $\frac{n}{d}$	Small

regular 每点连多个邻居

加边数 $m \approx \frac{k}{n}$ ↙边数 ↙点

优先依附 $P(v_i) = \frac{k_i}{\sum_j k_j}$ degree

$RC(P_i) = \frac{1}{2}$

$$EB(e) = \sum_{s<t} \frac{\sigma_{st}(e)}{\sigma_{st}}$$

$$\min_{\hat{X}} Tr(\hat{X}^T L \hat{X}) \quad s.t. \ \hat{X}^T \hat{X} = I$$

X打竖是一个vector

$$\hat{x}_i = \frac{x_i}{\sqrt{x_i^T x_i}}$$

x_{ij}表示i是否在j里

$L = D - A$

$I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$

$$cost_t = \alpha SC_t + (1-\alpha) TC_t$$

$$P = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN} \qquad F1 = \frac{2RP}{R+P}$$

$$Purity \ \frac{\sum_{i=1}^{k} \max_j |P_i \cap C_j|}{N}$$

$$\log\left(\frac{\exp(z_u^T z_v)}{\sum_{n\in V}\exp(z_u^T z_n)}\right) \approx \log\sigma(z_u^T z_v) + \sum_{i=1}^{k}\log\sigma(-z_u^T z_{n_i})$$

$L_{u,v} = -\log\sigma(z_u^T z_v) - \sum_{i=1}^{k}\log\sigma(-z_u^T z_{n_i})$ 负采样

$P(v|z_u) = \frac{\exp(z_u^T z_v)}{\sum_{v\in V}\exp(z_u^T z_v)}$ $L = \sum_{u\in V}\sum_{v\in N_R(u)} -\log\left(\frac{\exp(z_u^T z_v)}{\sum_{n\in V}\exp(z_u^T z_n)}\right)$

$p^{(t)}(Y_v = c) = \frac{\sum_{u\in N(v)} A_{vu}\, p^{(t-1)}(Y_u = c)}{\sum_{u\in N(v)} A_{vu}}$ 直到 $\forall v\ |p^{(t)}(Y_v) - p^{(t-1)}(Y_v)| \le \varepsilon$

$f: u \mapsto \mathbb{R}^d$ $f(u) = z_u$

$\arg\max_z \sum_{u\in V}\log P(N_R(u)|z_u) = \arg\min_z L = \sum_{u\in V}\sum_{v\in N(u)} -\log P(v|z_u)$

Unsupervised $L = -\sum_{u,v\in D}\log\sigma(z_v^T z_u) - \sum_{v,n\in D^-}\log\sigma(-z_v^T z_n)$

GCN $h_v^k = \sigma\left(W_k \sum_{u\in N(v)}\frac{h_u^{k-1}}{|N(v)|} + B_k h_v^{k-1}\right), \forall k \in \{1,\dots,K\}$

$h_v^0 = x_v$, $z_v = h_v^K$

Supervised $\hat{y}_v = \text{softmax}(W_{out} z_v + b_{out})$

CE $L = -\sum_{v\in V_{train}}\sum_{c=1}^{C} I[y_v = c]\log\hat{y}_{v,c}$

GCN aggregation weight is $|N(v)|$

GraphSAGE $h_v^k = \sigma([W_k \cdot AGG(\{h_u^{k-1}, \forall u\in N(v)\}), B_k h_v^{k-1}])$

$= \sigma(W^k \cdot CONCAT(h_v^{k-1}, AGG(\{h_u^{k-1}, \forall u\in N(v)\})))$

Pool $AGG = \gamma(\{Q h_u^{k-1}, \forall u\in N(v)\})$ element-wise mean/max

Graph Attention $e_{vu} = a(W_k h_u^{k-1}, W_k h_v^{k-1})$

$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{k\in N(v)}\exp(e_{vk})}$ $h_v^k = \sigma(\sum_{u\in N(v)} \alpha_{vu} W_k h_u^{k-1})$ 激活

一个能接触 $\beta N \cdot \frac{S}{N} \cdot I = \beta IS$ 1个人 易感比例

SI $\frac{dS}{dt} = -\beta IS$ $\frac{dI}{dt} = \beta IS$ $\frac{dI}{dt} = \beta NI - \beta I^2$ $I(t) = \frac{NI_0 e^{\beta N t}}{N + I_0(e^{\beta N t} - 1)}$

SIR $\frac{dS}{dt} = -\beta IS$ $\frac{dI}{dt} = \beta IS - \gamma I$ $\frac{dR}{dt} = \gamma I = \gamma(N - S - R)$ 增

$\beta S > \gamma$ 增 $\beta S < \gamma$ 减

$\frac{dS}{dR} = -\frac{\beta}{\gamma}S \Rightarrow S = S_0 e^{-\frac{\beta}{\gamma}R}$ 代入 $R_0 = 0$ $S(0) = S_0$

$t = \frac{1}{\gamma}\int_0^R \frac{dx}{N - S_0 e^{-\frac{\beta}{\gamma}R} - x}$ 无闭式解

~~SIR~~ SIS $\frac{dS}{dt} = \gamma I - \beta IS$ $\frac{dI}{dt} = \beta IS - \gamma I$

$\frac{dI}{dt} = I(\beta N - \gamma - \beta I^2)$ 若 $\beta N \leq \gamma$ $I(t) \downarrow \to 0$

若 $\beta N > \gamma$ $\Rightarrow$ $I(\beta N - \gamma) - \beta I^2 = 0$ $\begin{cases} I^* = 0 \\ I^* = N - \frac{\gamma}{\beta} \end{cases}$ logistic growth 非零稳定态

SIRS $\frac{dS}{dt} = \lambda R - \beta IS$ $\frac{dI}{dt} = \beta IS - \gamma I$ $\frac{dR}{dt} = \gamma I - \lambda R$ 无闭式解.数值计算

$f: U \times I \to R$

Content-Base 求 $sim(U_i, U_j) = \frac{\sum_{l=1}^{k} u_{il} i_{jl}}{\sqrt{\sum_{l=1}^{k} u_{i,l}^2} \sqrt{\sum_{l=1}^{k} i_{j,l}^2}}$ 其实就是归一化点乘

Memory-Based

① User-Based $r_{u,i} = \bar{r}_u + \frac{\sum_{u \in N(v)} sim(u,v)(r_{v,i} - \bar{r}_v)}{\sum_{u \in N(v)} sim(u,v)}$

cosine sim $Sim(U_x, U_v) = \frac{\sum_i r_{u,i} r_{v,i}}{\sqrt{\sum_i r_{u,i}^2} \sqrt{\sum_i r_{v,i}^2}}$

Pearson 减平均 $sim(U_x, U_v) = \frac{\sum_i (r_{u,i} - \bar{r}_u)(r_{v,i,} - \bar{r}_v)}{\sqrt{\sum_i (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_i (r_{v,i} - \bar{r}_v)^2}}$

② Item-Based $r_{u,i} = \bar{r}_i + \frac{\sum_{j \in N(i)} sim(i,j)(r_{u,j} - \bar{r}_j)}{\sum_{j \in N(i)} sim(i,j)}$

$X = U \Sigma V^T$ $X_k = U_k \Sigma_k V_k^T$

MAE $\frac{1}{n} \sum |p_i - r_i|$ $NMAE = \frac{MAE}{r_{max} - r_{min}}$ $RMSE = \sqrt{\frac{1}{n} \sum (p_i - r_i)^2}$

Precision $= \frac{\text{提+对}}{\text{提}}$ $Recall = \frac{\text{提+对}}{\text{对}}$ $F = \frac{2RP}{P+R}$

对是GT T是model

Spearman's rank $\rho = 1 - \frac{6 \sum_{i=1}^{n} (x_i - y_i)^2}{n^3 - n} \geq 1$

Kendall's T $T = \frac{c - d}{\binom{n}{2}}$ 正确-错误 总对数