8. Deep Graph Neural Networks for Recommender Systems

Summary: *In chapter 8,* ***we will study graph-based recommender systems****. We will present* ***node similarity measures based on the local structure of the graph*** *(e.g., adjacent nodes, common neighbors, preferential attachment, etc.). Moreover, we will describe algorithms such as* ***Random Walk with Restart*** *(****Personalized PageRank****),* ***SimRank****, and* ***PathSim****, which take under consideration the whole structure of the graph (e.g., the path length that connects two nodes of the graph, the number of different paths that connect two nodes, etc.). Furthermore, we will study* ***recommender systems based on Knowledge Graphs****, where data are stored with the help of an ontology, which allows the new knowledge inference based on simple* ***description logic rules****. Finally, we will describe in details the* ***Graph Convolutional Netwoks*** *and* ***Graph Embeddings****.*

Prerequisite Knowledge: *It is advised that the reader studies chapters 1 and 2 in advance.*

8.1 Introduction to Heterogeneous Graphs: Fundamentals

In this section we introduce the concept of *heterogeneous information network* by defining as a current example a *network schema* of an online newspaper, which will be used to illustrate the *graph-based similarity search algorithm* known as *Personalized PageRank* (or *Random Walk with Restart*).

8.1.1 Heterogeneous Information Network Definition

Definition 8.1 ***Information Network (Yizhou et al. [2011]).*** *An information network is defined as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a vertex type mapping function $\phi : \mathcal{V} \rightarrow \mathcal{Q}$ and a link type mapping function $\psi : \mathcal{E} \rightarrow \mathcal{R}$, where each vertex $v \in \mathcal{V}$ belongs to only one particular node type $\phi(v) \in \mathcal{Q}$, and each link $e \in \mathcal{E}$ belong to only one particular relation $\psi(e) \in \mathcal{R}$.*

Unlike the traditional definition of *information network*, as defined in Definition 8.1, we may clearly distinguish the different node types and the different edge types of the network. Thus, when

the node types $|\mathcal{Q}| > 1$ or the edge types $|\mathcal{R}| > 1$ are more than one, then the network is called a *Heterogeneous Information Network.*

Example 8.1 We are given a *heterogeneous information network* that represents the interaction of users with an online newspaper, and consists of nodes from five different types of entities $\mathcal{Q} = \{U, S, A, C, L\}$: users ($U$), sessions ($S$), articles ($A$), article categories ($C$), and geographic locations related to the articles (L). More precisely, each user $u : \phi(u) = U$ is connected by one or more edges to sessions $s : \phi(s) = S$. Each session s has a unique user u associated with it and one or more articles $a : \phi(u) = A$, which was read by the user within the session. Finally, each article a may appear in one or more sessions s, belongs to a news category $c : \phi(c) = C$ and is associated with a geographical location $l : \phi(l) = L$.

8.1.2 Network Schema Definition

Definition 8.2 ***Network Schema (Yizhou et al. [2011]).*** *The network schema is a meta template for a heterogeneous information network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the node type mapping $\phi : \mathcal{V} \rightarrow \mathcal{Q}$ and a link type mapping $\psi : \mathcal{E} \rightarrow \mathcal{R}$, which is a directed graph defined over node types $\mathcal{Q}$, with edges as relations from $\mathcal{R}$, denoted as $T_G = (\mathcal{Q}, \mathcal{R})$.*

Network schema serves as a generic template of the nodes' connections of the network, and tells how many types of objects there are in the network and where the possible links exist. An abstract network schema for our online news portal example is shown in Figure 8.1. A detailed representation of the Network Schema, defining also node attributes and relationship types is presented in Figure 8.2.

8.2 Related Bibliography

In this section we review related work on predicting future *links* between the *nodes* of a *graph.* For example, the research area of predicting links in *social networks* (e.g., friendship networks) attempts to make a prediction of which new interactions between members of a social network is likely to occur in the near future. There are two main research directions [Liben-Nowell and Kleinberg, 2003] that handle the *link prediction* problem, and these are briefly described below, but will be presented in detail later.

The first research direction, which is based on *local attributes* of a network, focuses mainly on the local link structure of nodes. There is a variety of *local similarity* measures [Liben-Nowell and Kleinberg, 2003], such as Adamic and Adar's *similarity index* [Adamic and Adar, 2005], the *Friend of a Friend (FOAF)* algorithm [Chen et al., 2009], the *preferential attachment algorithm* [Liben-Nowell and Kleinberg, 2003], etc.

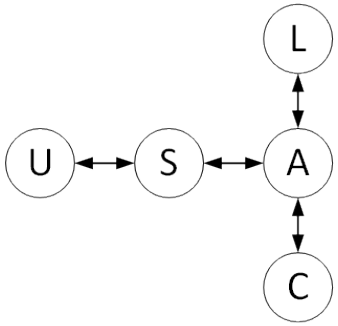


Figure 8.1: Online News portal Network Schema (abstract)

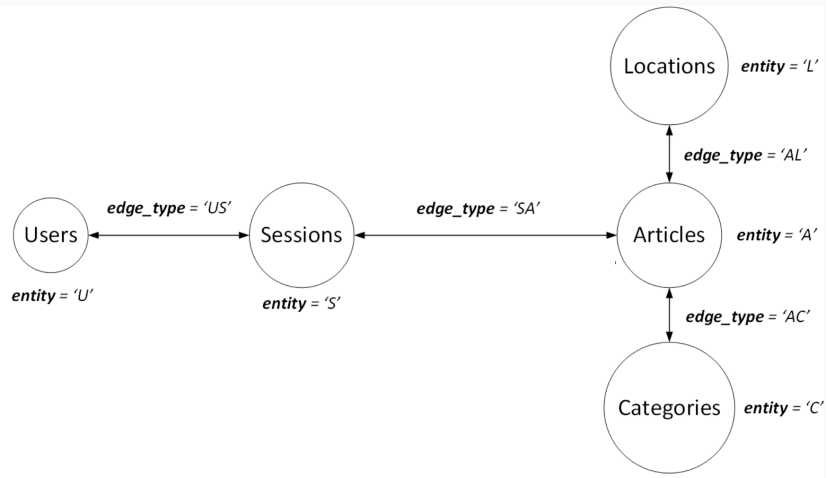


Figure 8.2: Online News portal Network Schema (detailed)

The second research direction is based on the *global attributes* of a graph, since it detects the entire structure of the graph. A variety of approaches exist here as well, such as the *Random Walk with Restart* algorithm [Pan et al., 2004], the *SimRank algorithm* [Jeh and Widom, 2002], the *Katz algorithm* [kat], the *PathSim algorithm* [Sun et al., 2011], etc. All the aforementioned algorithms will be described in detail in the following subsections.

Besides the above *link prediction algorithms*, which are mainly applied to *unipartite graphs*, there are other methods that exploit additional data sources, such as messages between users, co-authors of a paper, and *shared tagging of information objects* (*tags* in photos, videos, etc.). For example, [Guy et al., 2009] proposed a new *widget* for providing a list of recommended friends to the target user. This list, then, was based on aggregated information collected from various sources across the *IBM* organization. Finally, the work of Chen and co-authors [Chen et al., 2009] evaluated four recommendation algorithms (*Content Matching, Content-plus-Link, FOAF* and *SONAR*) that aim to help users to discover new friends in *IBM*'s online social network.

8.3 Local-based Similarity Measures

8.3.1 Shortest Path

Shortest path-based similarity measure: The *shortest path measure* calculates the shortest distance between any pair of nodes/users in a social network. Therefore, the nearest nodes can be suggested to the target user v_x, and to achieve this we usually use the well-known *Dijkstra's algorithm* or a more efficient algorithm [Fredman and Tarjan, 1987].

8.3.2 Common Neighbors

Common neighbors similarity measure: The *Common Neighbors (CN)* similarity measure or else known as *Friend of a Friend*[Chen et al., 2009] algorithm is based on the number of friends that the two target nodes v_x and v_y have in common according to Equation 8.1:

$$score(v_x, v_y) := |\Gamma(v_x) \cap \Gamma(v_y)| \tag{8.1}$$

where $score(v_x, v_y)$ is the degree of relevance of v_x and v_y, and $\Gamma(v_x), \Gamma(v_y)$ are the sets of neighboring nodes of v_x and v_y respectively. Thus, $|\Gamma(v_x) \cap \Gamma(v_y)|$ is the number of their joint neighbor nodes. The candidate friends are proposed to v_x in descending order based on their *score*.

8.3.3 Jaccard similarity index

Jaccard index: Jaccard index is very similar to the *common neighbor similarity measure*. More specifically, it captures the relevance between two nodes of a graph by measuring the degree of overlap between their neighboring nodes, and is defined as follows:

$$score(v_x, v_y) = \frac{|\Gamma(v_x) \cap \Gamma(v_y)|}{|\Gamma(v_x) \cup \Gamma(v_y)|}$$

where in the numerator of the above equation we compute the number of common neighbors of the two nodes under consideration, while in the denominator we compute the sum of their neighbours.

8.3.4 Salton similarity index

The Salton similarity index [Chowdhury, 2010]: This index can be seen as an improved version of the *index of common neighbours* because, in addition to the number of common neighbours, it takes into account the *degree* of the nodes under consideration. It is defined as follows:

$$score(v_x, v_y) = \frac{|\Gamma(v_x) \cap \Gamma(v_y)|}{\sqrt{d_{v_x} * d_{v_y}}}$$

where d_{v_x} and d_{v_y} are the degree of v_x and v_y respectively.

8.3.5 Adamic & Adar similarity index

Adamic & Adar similarity index [Adamic and Adar, 2005] proposed a *distance measure* to determine when two web pages are relevant. That is, they calculated attributes of the web pages and defined the similarity between two pages x, y as follows: $\sum_z \frac{1}{\log(frequency(z))}$ where z is an attribute shared by pages x, y. This concept improves the simple count of shared features by weighting the rarer features more heavily. The similarity between nodes v_x and v_y can be computed from Equation 8.2:

$$score(v_x, v_y) = \sum_{z \in \Gamma(v_x) \cap \Gamma(v_y)} \frac{1}{\log |\Gamma(z)|} \tag{8.2}$$

where $\Gamma(v_x), \Gamma(v_y)$ are the common neighbors of v_x and v_y respectively.

8.3.6 Preferential Attachement

Preferential Attachment: The idea here is that the probability of a new edge being attached to the target node v_x is proportional to its node degree. In other words, the above probability is proportional to the number of its neighboring nodes. Barabasi et al. [Barabasi et al., 2002] and Newman [Newman, 2001] proved empirically that the probability of a new edge being connected to nodes v_x and v_y is correlated with the product of degrees of the nodes v_x and v_y, as defined in Equation 8.3:

$$score(v_x, v_y) := |\Gamma(v_x) \cdot \Gamma(v_y)| \tag{8.3}$$

where $\Gamma(v_x), \Gamma(v_y)$ are the sets of neighboring nodes of v_x and v_y respectively.

8.3.7 FriendLink

FriendLink algorithm: The *similarity index FriendLink* [Papadimitriou et al., 2011] is used for the *link prediction problem* between nodes of a graph. In online social networks such as *Facebook*, users explicitly declare their friends so that they are able to share information with them (e.g. photos, videos, etc.). The FriendLink similarity measure assumes that individuals in a friendship graph can use all paths that connect them. Each path contributes to the similarity according to its length. Thus, two users who are connected with many paths have a higher probability of knowing each other in proportion to the length of those paths (i.e., the shorter the path length, the greater the probability of affinity). The *similarity* $score(v_x, v_y)$ between two nodes v_x and v_y of a graph is defined as the number of paths of different lengths ℓ from v_x to v_x in v_y:

$$score(v_x, v_y) = \sum_{i=2}^{\ell} \frac{1}{i-1} \cdot \frac{\left|paths^i_{v_x, v_y}\right|}{\prod_{j=2}^{i}(n-j)} \tag{8.4}$$

where:

- n is the number of nodes of a graph G,
- ℓ is the maximum length of a path connecting nodes v_x and v_y (excluding *cyclic paths*). By *cyclic paths* we mean paths that are *closed*. A node may appear once in a path, while the path $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_1 \rightarrow v_5$ is *cyclic/closed*, because v_1 appears twice,
- $\frac{1}{i-1}$ is an attenuation factor of the importance of long paths, which weights the paths according to the length ℓ each has. Thus, a path of length 2 has a value equal to 1 ($\frac{1}{2-1} = 1$), while a path of length 3 has a value equal to $\frac{1}{2}$ ($\frac{1}{3-1} = \frac{1}{2}$), etc. In this sense, we appropriately use a *weighting scheme* to reduce the importance of the longer paths,
- $\left|paths^{\ell}_{v_x,v_y}\right|$ the number of all paths of length-ℓ from v_x to v_y,
- $\prod_{j=2}^{i}(n-j)$ the number of all possible paths of length ℓ from v_x to v_y, if each node in the graph G was connected to all other nodes. Using the fraction $\frac{\left|paths^{\ell}_{v_x,v_y}\right|}{\prod_{j=2}^{i}(n-j)}$ the similarity measure is normalized and takes values in [0,1].

So according to *similarity measure FriendLink*, if two nodes are highly relevant to each other, then we expect the value $score(v_x, v_y)$ to be close to 1. On the other hand, if two nodes are not relevant, we expect the value $score(v_i, v_j)$ to be close to 0.

8.4 Global-based Similarity Algorithms

8.4.1 Katz Status Index

To identify the most important nodes in a graph, Leo Katz [Katz, 1953] combined sociological theory with linear algebra by proposing the *Katz Status index*, which captures the "prestige" or status of a node in the graph. Next, we will use an example to show how the aforementioned status index can be modified to compute the *similarity* between the nodes of a graph.

Example 8.2 We are given the *friendship graph* G of Figure 8.3, which represents the users of a social network. Regarding the basic characteristics of graph G, it consists of a set of nodes V and a set of edges E. Furthermore, each edge is defined by a specific pair nodes of the graph (v_i, v_j) where $v_i, v_j \in V$. Also, we assume that the graph G is *non-directed* and *unweighted*. This means that the edges of the graph have no weights, and that the order of nodes on an edge is not important. Therefore (v_i, v_j) and (v_j, v_i) denote the same edge in G. We assume, in addition, that the graph G cannot have multiple edges connecting two nodes. Therefore, if two nodes v_i, v_j are connected to an edge of E, then there cannot be another edge in E that

connects them, given that E is the set of edges of our graph. Finally, we assume that there can be no loop edges in the graph G (i.e. a node cannot be connected to itself via an edge). For the graph of Figure 8.3, we are asked to compute the *similarity* of the nodes of the graph using the *Katz status index*.

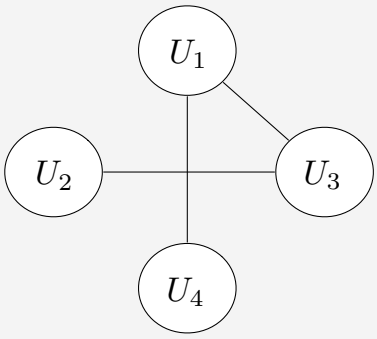


Figure 8.3: A friendship graph, which represents users who are connected to each other with "friendship" edges.

A graph G is usually represented by means of an adjacency matrix A of dimension $n \times n$, where $n = |V|$ is the number of nodes of G. In other words, the adjacency matrix has n rows and n columns. For the graph of Figure 8.3, the entries of the adjacency matrix are defined as follows:

$$A[v_i, v_j] = \begin{cases} 1, & \text{If } (v_i, v_j) \in E \\ \\ 0, & \text{If } (v_i, v_j) \notin E \end{cases}$$

Note that the *adjacency matrix* of an *undirected and unweighted graph* G is a *symmetric matrix* with values 1 and 0 respectively, showing if two nodes are adjacent or not. Moreover, since there are no *loop edges* (*self-connected edges*), the principal diagonal of the *adjacency matrix* has zero values. Consequently, the *adjacency matrix* of the graph for the data of Example 8.2 is shown in Table 8.1.

	U_1	U_2	U_3	U_4
U_1	0	0	1	1
U_2	0	0	1	?
U_3	1	1	0	?
U_4	1	?	?	0

Table 8.1: *Adjacency matrix A* for the data of Example 8.2.

As depicted in Table 8.1, U_1 is connected to U_3 and U_4, while U_2 is only connected to U_3. Also,

U_1 and U_2 have a common friend U_3, since they are both connected to this user. So suppose that for the social network of Figure 8.3, we want to recommend new friends to user U_4. Thus, as we want to predict the *unknown values* (see the question marks - ?) of Table 8.1, we may assume that they are initially equal to 0 (i.e., there are no edges between the respective users). We will first present the theoretical framework and then give a solution for the data in Example 8.2.

There are, of course, several similarity measures (e.g., *Katz Status index*, *RWR algorithm*, *SimRank algorithm*, etc.) that take into account the *total graph's structure* [Liben-Nowell and Kleinberg, 2003] to capture the similarity between pair of nodes, which may depend on the number of paths connecting them. We adapt correspondingly the *Katz Status index*, to capture the *similarity score* between two nodes V_x and V_y by aggregating the paths of different lengths ℓ connecting V_x to V_y, as illustrated in the following Eq. 8.5:

$$Katz_\beta = \sum_{\ell=1}^{\infty} \beta^\ell |paths^\ell_{V_x,V_y}| \tag{8.5}$$

where $paths^\ell_{V_x,V_y}$ is the set of all paths of length-ℓ from node V_x to V_y, computed from the *adjacency matrix A*. The *Katz Status index* exploits the fact that raising the *adjacency matrix* to the power n produces the number of length-n paths that connect a pair of nodes. Also, in Equation 8.5 there is a *damping coefficient* β (or else, attenuation factor β), which weights the paths of different lengths according to their contribution to the final *similarity score* of the nodes. This coefficient can therefore take values such as $\beta < 1/\lambda$, where λ is the largest eigenvalue of the *adjacency matrix* A [Katz, 1953, Foster and Muth, 2002]. Very small values of β lead to higher weights of the closest paths. This fact makes the overall score function focus more on the *nearest neighbours of the target node*, thus making it more efficient. On the other hand, very low values of β result in very long paths contributing much less to the overall *similarity* between two nodes of the graph. Please note that the following Equation 8.6 is the analytical form of the *Katz Status index* when applied to the *adjacency matrix A*:

$$Katz(\mathbf{A};\beta) = \beta\mathbf{A} + \beta^2\mathbf{A}^2 + \beta^3\mathbf{A}^3 + ... = \sum_{\ell=1}^{\infty} \beta^\ell \mathbf{A}^\ell, \tag{8.6}$$

which is also expressed with the help of linear algebra matrices as follows:

$$\sum_{\ell=1}^{\infty} \beta^\ell \mathbf{A}^\ell = (I - \beta\mathbf{A})^{-1} - I \tag{8.7}$$

where the *identity matrix I* is a square matrix of dimension $n \times n$, which contains units on its principal diagonal and has the same size n as the *adjacency matrix A*. And the *attenuation factor* β is a parameter that ensures the convergence of the above sequence and allow the computation of the inverse of the *adjacency matrix A*. We choose β equal to $1/K$ -as Foster and Muth do [Foster and Muth, 2002]- and where K is the largest sum of rows or columns of the adjacency matrix A.

In the current example, then, suppose we compute the similarity between U_4 and U_2 and U_3 respectively. We therefore apply the *Katz status index* to the *friendship graph* G in order to provide recommendations based on the computation of a *similarity matrix* among the nodes of the graph. More specifically, we compute the *Katz index* by applying the Equation 8.7 to the *adjacency matrix* A of the Table 8.1. The *Katz index* calculates the similarity between two nodes considering only paths of length $\ell > 1$.

	U_1	U_2	U_3	U_4
U_1	0	0.1636	0.4909	0.4364
U_2	0.1636	0	0.4364	0.0545
U_3	0.4909	0.4364	0	0.1636
U_4	0.4364	**0.0545**	**0.1636**	0

Table 8.2: *The user-user similarity/similarity table* we compute for the data in Example 8.2.

Notice that the similarity between U_4 and U_2 is computed based on the unique path connecting them (4⟶1⟶3⟶2), as shown in Figure 8.3. According to the user similarity predictions of Table 8.2, the 3-step-long path contributes a *score similarity* equal to 0.0545. Regarding the similarity between U_4 and U_3, there is only one 2-hop long path (4⟶1⟶3)-as shown in Figure 8.3-which contributes a *score similarity* equal to 0.1636.

The *user-user similarity scores* of Table 8.2 capture the future friendship relations we predict for the social network of Example 8.2. There is a clear indication from the above *similarity matrix* that U_4 should be recommended as a friend to U_3 over U_2, since U_4 has with U_3 a higher *similarity score*, $0.1636 > 0.0545$.

8.4.2 SimRank for Unipartite Graphs

Jeh and Widom [Jeh and Widom, 2002] proposed in 2002 a *similarity measure* of the nodes in a graph known as *SimRank*, which can be used to measure similarities either between nodes of the same type (e.g. similarities between users considering their friendship relations) or between different types of nodes (e.g. similarities between users considering the items they interact with). The idea of *SimRank* is as follows:

Two nodes of a graph are ***similar*** *if they are referenced by* ***similar*** *nodes.*

For example, two articles are *relevant* if they are read by *relevant* users.

Definition 8.3 ***SimRank.*** *The similarity between two nodes a and b of a unipartite directed graph G is defined as follows:*

$$s(a,b) = \begin{cases} 1, & \text{Αν } a = b; \\ \frac{C}{|I(a)||I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} s(I_i(a), I_j(b)), & \text{Αν } a \neq b, \text{ και } (I(a) \neq \varnothing \text{ ή } I(b) \neq \varnothing); \\ 0, & \text{Αν } I(a) = \varnothing \text{ ή } I(b) = \varnothing, \end{cases} \tag{8.8}$$

*where $C \in [0,1]$ is the **attenuation coefficient** or else, **damping factor**, $I(a)$ are the incoming neighbor nodes of a, and $I(b)$ are the incoming neighbor nodes of b.*

Based on Equation 8.8 to compute the *similarity* $s(a,b)$ of nodes a and b, we cumulatively sum the *similarity* $s(I_i(a), I_j(b))$ obtained from all possible pairs of their *in-coming* neighboring nodes $(I_i(a), I_j(b))$. Then we divide by the total number of possible pairs of *in-coming neighboring nodes* $|I(a)||I(b)|$, to normalize the computed cumulative similarity. In other words, the *similarity* between nodes a and b is the *average similarity* between the incoming neighbors of a and the incoming neighbors of b. Equation 8.8 is computed iteratively and usually converges quickly (after the fourth or fifth iteration).

Example 8.3 We are given the *unipartite and directed graph* of Figure 8.4, whose nodes are news articles that reference each other. We are asked to: (a) Apply Equation 8.8 to compute the *similarity score* between articles A4 and A5, as well as between articles A1 and A5 after only the first iteration of the *SimRank algorithm*. (b) Compute the *similarity score* between all articles of the graph by repeatedly running the *SimRank algorithm* until it converges. The input values of the parameters of the SimRank algorithm are $C = 0.8$ and $C = 0.8$ and $\varepsilon \leqslant 10^{-4}$. Notice in Figure 8.4 that the *incoming neighbor nodes*, which reference the target article A4 are articles A1, A2, και A3. Also, the *incoming neighbor node*, which references the target article A5 is article A3. We note here that both nodes A4 and A5 jointly have the incoming neighbor article A3 referencing them. Based on the above data, with Equation 8.8 we can first compute the cumulative similarity resulting from all possible combinations of *incoming neighbor nodes* that reference at least one of the two target nodes:

$$\sum_{i=1}^{|I(A4)|} \sum_{j=1}^{|I(A5)|} s\left(I_i(A4), I_j(A5)\right) = s(A1,A3) + s(A2,A3) + s(A3,A3) = 0 + 0 + 1 = 1.$$

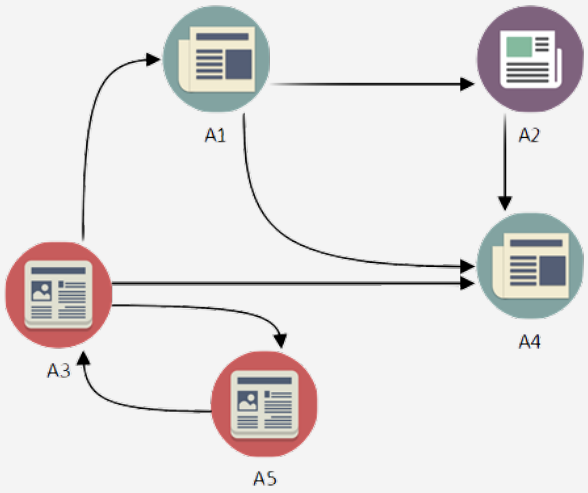


Figure 8.4: A *single graph* with news articles.

Finally, applying the *normalized attenuation coefficient* $\frac{C}{|I(A4)||I(A5)|}$ of Equation 8.8 the *similarity* between articles A4 and A5 ends up being the following:

$$s(A4, A5) = \frac{0.8}{3 * 1} * 1 = 0.267.$$

Thus, the *affinity* between articles $A4$ and $A5$ is equal to 0.267.

Notice in Figure 8.4 that articles A1 and A5 have only one common *incoming neighbor node* (the news article A3) that both references them. Thus, their *similarity* is computed as follows:

$$s(A1, A5) = \frac{0.8}{1 * 1} * 1 = 0.8.$$

Therefore, the *similarity* between articles $A1$ and $A5$ is larger than that between articles A4 and A5 that we had computed earlier, because A4 has 3 *in-coming neighbor nodes*, while A1 has only 1 *in-coming neighbor node*.

In the same direction, we apply the *SimRank* algorithm on all the data of Example 8.3 to compute the *similarity* among all nodes of the graph. The results of *relevance* among the articles are shown in Table 8.3. Looking at Figure 8.4 gives us the intuition that articles $A3$ and $A5$ should be more *relevant*, because they reference to each other. However, the *SimRank algorithm* cannot capture this intuition, because it only takes into account the *common incoming neighbor nodes*, which reference the two target nodes. Thus, because in our example news articles $A3$ and $A5$ have no *common incoming neighbor node*, we are led to compute zero *similarity* between them, as shown in Table 8.3. Of course, this is a disadvantage of the *SimRank algorithm*.

	A1	A2	A3	A4	A5
A1	1.000	0.000	0.000	0.437	0.800
A2	0.000	1.000	0.640	0.267	0.000
A3	0.000	0.640	1.000	0.213	0.000
A4	0.437	0.267	0.213	1.000	0.437
A5	0.800	0.000	0.000	0.437	1.000

Table 8.3: *Similarity matrix* among all news articles, which is computed by using the *SimRank algorithm* applied on the data of Example 8.3.

8.4.3 SimRank Algorithm for Bipartite Graphs

In *bipartite graphs* we have two different types of nodes (e.g. users and news article). The idea of *SimRank* for *bipartite graphs* is similar for *unipartite*. That is:

*Two news articles are **relevant** if they have been read by **relevant** users.*
*Two users are **relevant** if they have read **relevant** articles.*

As described before, Equation 8.8, concerns only unipartite graphs, and finds the *similarity* between articles (i.e., only the *incoming neighbor nodes* matter). Now, for a *user-news article bipartite graph*, the only difference in logic is that the *incoming neighbor nodes* no longer represent references from other news articles, but the reading behavior of users. Also, for the calculation of *similarity* between users (*user-user similarity*), *incoming nodes* are replaced by *out-going neighbor nodes*. Note that instead of always checking the type of node (i.e.,, users or news articles), it suffices to consider only the existence of a in-coming or out-going *link*. Please note that since each link in a *bipartite user-news article graph* only ever connects users to news articles, this means that each link can be treated as *outgoing* for a user and as *incoming* for an article.

Definition 8.4 ***SimRank for bipartite graphs.*** *The similarity between two nodes e and f of a bipartite graph G is defined as follows:*

$$s(e,f) = \begin{cases} 1, & \text{Αν } e = f \\ \frac{C}{|N(e)||N(f)|} \sum_{i=1}^{|N(e)|} \sum_{j=1}^{|N(f)|} s(N_i(e), N_j(f)), & \text{Αν } e \neq f \text{ και } (N(e) \neq \varnothing \text{ ή } N(f) \neq \varnothing) \\ 0, & \text{if } N(e) = \varnothing \text{ ή } N(f) = \varnothing, \end{cases} \tag{8.9}$$

where $N(e)$ and $N(f)$ are the sets of adjacent nodes of e and f respectively, and $C \in [0,1]$. That is, if a node is of type "user", its outgoing neighbors are the news articles with which it has interacted. And by analogy, if a node is of type "news article", then its incoming neighbors are

the users that have interacted with it. In other words, in *bipartite graphs* we do not consider only *incoming neighbor nodes*, but simply the existence of *neighbor nodes*, which may be either *in-coming* to the target node or *out-going* from the target node.

Example 8.4 We are given the *bipartite user-news article graph* of Figure 8.5, which displays the online newspaper articles that users have interacted with.

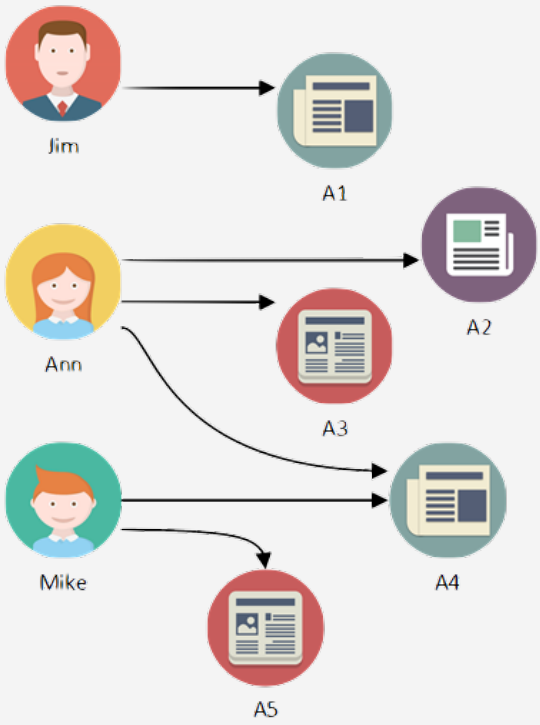


Figure 8.5: Example of a bipartite user-news article network

We are asked to compute the pairwise *similarity* among all news articles and between all users of the *bipartite graph* by iteratively running the *SimRank algorithm* until it converges. The input values of the parameters of the *SimRank* algorithm are the following: *atenuation factor* $C = 0.8$ and *convergence criteria* $\varepsilon \leqslant 10^{-4}$.

Next, we apply the *SimRank* algorithm on the data of Example 8.4 to compute the *similarity* between all nodes of the graph. The *similarity* score between news articles and between users are shown in Figure 8.6. Notice in the similarity matrix of Figure 8.6 that the *similarity* between news articles $A2$ and $A3$ is 0.8, because they are read by the same user, namely Ann: $\frac{0.8}{1*1} * 1$.

	Jim	Ann	Mike	A1	A2	A3	A4	A5
Jim	1.0	0.000	0.000	0.0	0.000	0.000	0.000	0.000
Ann	0.0	1.000	0.468	0.0	0.000	0.000	0.000	0.000
Mike	0.0	0.468	1.000	0.0	0.000	0.000	0.000	0.000
A1	0.0	0.000	0.000	1.0	0.000	0.000	0.000	0.000
A2	0.0	0.000	0.000	0.0	1.000	0.800	0.587	0.374
A3	0.0	0.000	0.000	0.0	0.800	1.000	0.587	0.374
A4	0.0	0.000	0.000	0.0	0.587	0.587	1.000	0.587
A5	0.0	0.000	0.000	0.0	0.374	0.374	0.587	1.000

Figure 8.6: *Similarity matrix* after applying the *SimRank algorithm* for the data of Example 8.4.

Please also note in the similarity matrix of Figure 8.6 that Ann and Mike have a higher *similarity score* compared to how similar they both are to Jim, because they have read a common article, A4. Finally, news article A2 is *similar* to news article A5 because they have both been read by *similar* users, namely Ann and Mike.

8.4.4 Random Walk with Restart algorithm

The *Random Walk with Restart (RWR)* [Pan et al., 2004, Tong et al., 2006] is a variant of the well-known *PageRank* algorithm. The *RWR* algorithm has properties that can adequately capture the notion of *similarity* between the target node N and the other nodes of a graph. The main advantage of *RWR* over *PageRank* is the "teleportation" feature, which forces the random "walker" to restart its "walk" from the starting node N, whereas in *PageRank* the random "walker" jumps randomly to any node in the graph. As expected, *RWR* assigns more *similarity* to nearby nodes of the starting node N. This means that if two nodes of the graph are close to each other, the probability that they will be connected via an edge in the future is higher. In social networks, *RWR* can capture the notion of *similarity* between users who share a large number of common friends. For a *bipartite user-item graph*, consequently, when two users interact with the same items, their overall probability of being connected (via a friendship node) increases.

Next, we study the *RWR algorithm* and its application to *heterogeneous graphs*, that consist of different node or edge types. We can, therefore, imagine a "random walker" which can jump from any node v_i to any of its neighbors with equal probability, and at each iteration step with some certain probability can return back from his random walk to the starting node N. In this way we want to determine where the "random walker" will end up after taking short "walks" starting from the target node N. Thus, we want to predict which nodes of the graph are most *relevant* to the target node N. This similarity measure is referred to as **Personalized PageRank** [Page and Brin,

1998], **Topic-Sensitive PageRank** [Haveliwala, 2002] or as **Random Walk with Restart** (*RWR*) [Leskovec et al., 2019].

After running the *RWR algorithm* on a graph with n nodes, we obtain as a result a similarity matrix $\mathbf{V}$ of dimension $n \times n$, where each element v_{ij} corresponds to the degree of *similarity* of a node i to node j. Unlike the similarity matrix computed by the *SimRank algorithm*, the similarity matrix obtained by the *RWR algorithm* is not symmetric. This means that the *similarity* of a node i to node j is not equal to the *similarity* of node j to node i. Therefore, the "random walker" is more likely to end up at a "well-connected" node than at a "non well-connected node".

Let us denote the *transition probability matrix* of a *unipartite graph* G with the symbol M. That is, the entry in row i and column j of M is equal to $1/k$ where k is the *degree* of node j (*node's degree*), and node i is one of its neighbors. If $k = 0$, then this entry equals 0. We emphasize that the *transition probability matrix* $\mathbf{M}$ is obtained by multiplying the *adjacency matrix* $\mathbf{A}$ by the inverse of the node degree matrix D^{-1} of our graph. In addition, suppose that we define by the variable β the probability for a "random walker" to randomly continue its "walk", and hence by $1-\beta$ the probability to interrupt his "random walk" and jump to the starting node N. Let, further, $\mathrm{e_N}$ be a column vector with 1 in the row that concerns node N, and 0 at any other position. Finally, let v be a column vector reflecting the probability that the "walker" is at a node at a particular time point t, and v' be the probability that the "random walker" is at a node at the next time point (i.e. $t+1$). Then the vector v' is related to v as follows:

$$\mathrm{v}' = \beta \cdot \mathrm{M} \cdot \mathrm{v} + (1-\beta) \cdot \mathrm{e_N} \tag{8.10}$$

Next, we define Equation 8.11 which transforms initial Equation 8.10, so that it can be applied to all nodes of graph G (rather than just one node of the graph), and compute a relevance matrix that holds the similarities between the nodes of the graph by using the *power iteration* method.

Definition 8.5 ***Random Walk with Restart (RWR).*** *The RWR method specifies a nonsymmetric and row-normalized relevance matrix* V' *where the sum of the per-row probabilities equals unity. The matrix* V' *stores the transition probabilities between any two nodes of the graph and can be computed iteratively using Equation 8.11.*

$$\mathrm{V}' = \beta \cdot \mathrm{M} \cdot \mathrm{V} + (1-\beta) \cdot \mathrm{I_n} \tag{8.11}$$

where $\mathrm{I_n}$ *is the identity matrix of dimension* $n \times n$*, whereas* V *and* V' *are the normalized transition probability matrices for two consecutive time points (e.g.* t *and* $t+1$*).*

Example 8.5 Given the *heterogeneous graph* of Figure 8.7, which consists of 4 different types of nodes (users, sessions, articles, article categories), we are asked to compute the *nodes' similarity* by iteratively running the *RWR algorithm* until it converges.

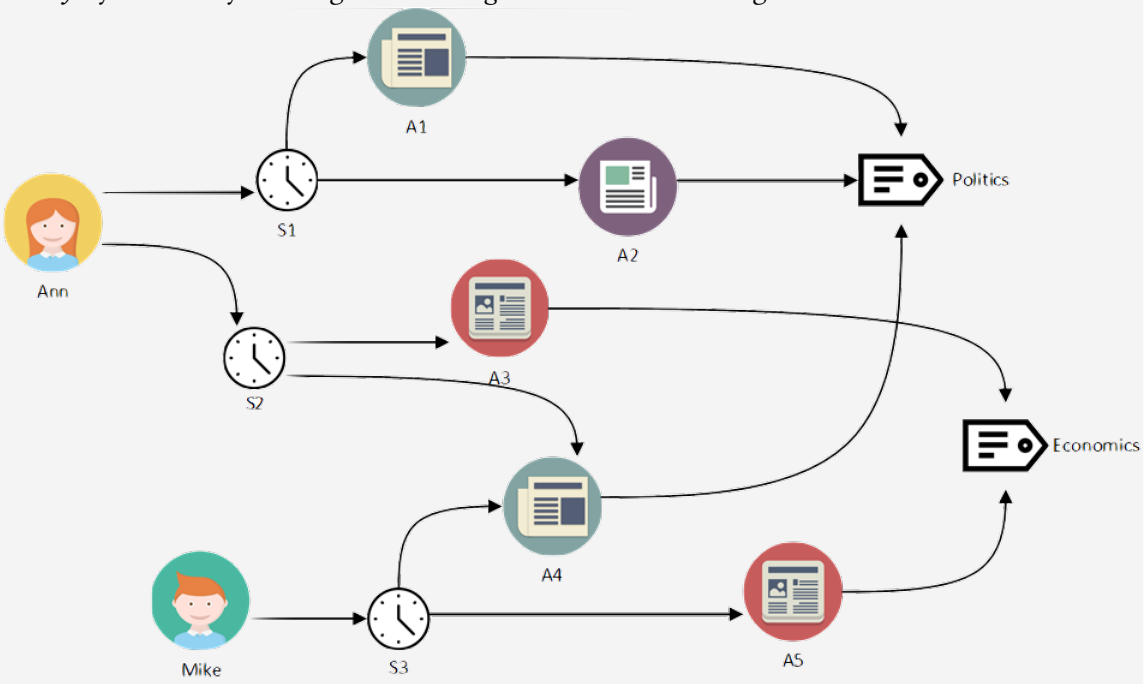


Figure 8.7: Heterogeneous graph of an online newspaper.

After applying the *RWR algorithm* to the data of Example 8.5, the results are shown in Figure 8.8. The main advantage of the similarity matrix is that it consists of sub-matrices, which are useful for various purposes. For example, the *Article-Article* sub-matrix (in the diagonal in Figure 8.8), shows the *transition probabilities matrix* from one article to another article, and can be used to *predict the user's next click* based on the article with which he or she is currently interacting.

The *User-Article Category* sub-matrix (top right corner in Figure 8.8), moreover, reveals the general preferences of users, determining how likely it is for the user to read an article of one or another article category (e.g., Politics or Economics). Please note also that an interesting and not so obvious observation appears in the *User-Article Category sub-matrix* of Figure 8.8. In particular, we can see from Figure 8.7 that Mike has read one article from each category (Politics and Economics). However, it is clear from the *relevance matrix* of Figure 8.8 that his interest in Economics is much higher than in Politics. The reason is that the A4 article that Mike read in session S3 appears in session S2 of Ann along with A3, which is an Economics article, thus bringing A4 closer to the Economics category, and thus, we predict an increasing interest of Mike in Economics. Finally, please note that Example 8.5 will be solved as a programming exercise with *Python* at the end of this chapter.

	Ann	Mike	S1	S2	S3	A1	A2	A3	A4	A5	Politics	Economics
Ann	0.2841	0.0089	0.1625	0.1525	0.0332	0.0612	0.0612	0.0524	0.0674	0.0206	0.0669	0.0291
Mike	0.0178	0.2764	0.0196	0.0469	0.2865	0.0157	0.0157	0.0335	0.0994	0.0974	0.0389	0.0523
S1	0.1083	0.0065	0.3393	0.0672	0.0245	0.1198	0.1198	0.0236	0.0538	0.0123	0.1103	0.0144
S2	0.1016	0.0156	0.0672	0.3141	0.0588	0.0331	0.0331	0.1071	0.1146	0.0390	0.0571	0.0585
S3	0.0222	0.0955	0.0245	0.0588	0.3583	0.0195	0.0195	0.0418	0.1241	0.1216	0.0488	0.0654
A1	0.0612	0.0078	0.1797	0.0497	0.0292	0.2942	0.0942	0.0182	0.0673	0.0128	0.1732	0.0124
A2	0.0612	0.0078	0.1797	0.0497	0.0292	0.0942	0.2942	0.0182	0.0673	0.0128	0.1732	0.0124
A3	0.0524	0.0167	0.0355	0.1607	0.0627	0.0182	0.0182	0.3040	0.0684	0.0778	0.0327	0.1527
A4	0.0450	0.0331	0.0538	0.1146	0.1241	0.0449	0.0449	0.0456	0.2942	0.0481	0.1143	0.0374
A5	0.0206	0.0487	0.0184	0.0585	0.1825	0.0128	0.0128	0.0778	0.0722	0.3109	0.0294	0.1554
Politics	0.0446	0.0130	0.1103	0.0571	0.0488	0.1155	0.1155	0.0218	0.1143	0.0196	0.3229	0.0166
Economics	0.0291	0.0261	0.0216	0.0878	0.0981	0.0124	0.0124	0.1527	0.0561	0.1554	0.0249	0.3233

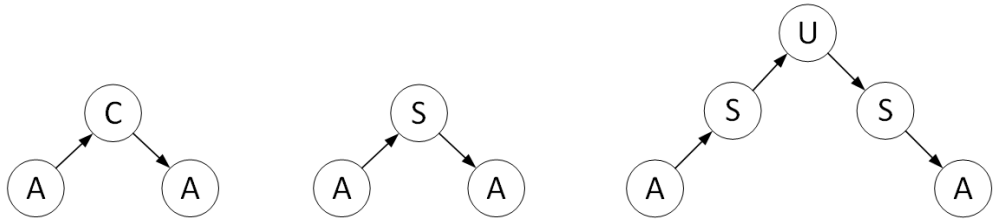
Figure 8.8: *Relevance matrix* among nodes of Example 8.5.

8.4.5 PathSim Algorithm

Yizhou et al. [Yizhou et al., 2011] proposed a new concept for measuring the *similarity* between nodes in a network, which is based on the analysis of the *meta paths* through which nodes are connected. In a heterogeneous network two nodes may be connected via different paths. For example, two articles can be connected via the article-category-article path (*similarity based on content*), article-session-article path (*similarity based on session*) and article-session-user-session-article path (*similarity based on collaborative filtering*). Using different paths, different degrees of *similarities* are observed. These paths are called *meta paths* as mentioned above and are typically defined as follows:

Definition 8.6 ***Meta Path.*** *[Yizhou et al., 2011] A meta-path $\mathcal{P}$ is a path defined in the graph of the network schema $T_G = (\mathcal{Q}, \mathcal{R})$, and is denoted in the form $Q_1 \xrightarrow{R_1} Q_2 \xrightarrow{R_2} \ldots \xrightarrow{R_l} Q_{l+1}$, which defines a complex relation $R = R_1 \circ R_2 \circ \ldots \circ R_l$ between different types of nodes Q_1 και Q_{l+1}, where $\circ$ denotes the composition operator of the connections/links, or edges.*

There are many different *meta-paths* that can be found in the graph of Figure 8.7, whose *network schema* is USAC (User, Session, Article, and Category). If we start from an "article" node, we can then construct the following *meta paths* ACA, ASA, ASUSA, as shown in Figure 8.9. The main advantage of the theoretical framework of *meta-paths* is its *explanability* and thus accountability. That is, it provides a powerful mechanism for the user to choose the appropriate semantics that will determine the *similarity* between nodes by choosing an appropriate *meta-path.* **PathSim** is a *similarity measure* that is therefore able to capture the semantics of the *similarity* between nodes in a network.



(a) *Meta Path*: ACA (b) *Meta Path*: ASA (c) *Meta Path*: ASUSA

Figure 8.9: Possible *meta-paths* of the USAC network.

Definition 8.7 ***PathSim: A similarity measure based on Meta paths.*** *[Yizhou et al., 2011] Given a symmetric meta-path P, the similarity measure PathSim between two nodes of the same type x and y is:*

$$s(x,y) = \frac{2 * |p_{x \rightsquigarrow y} : p_{x \rightsquigarrow y} \in P|}{|p_{x \rightsquigarrow x} : p_{x \rightsquigarrow x} \in P| + |p_{y \rightsquigarrow y} : p_{y \rightsquigarrow y} \in P|} \tag{8.12}$$

where $p_{x \rightsquigarrow y}$ is a path that starts at the origin node x and ends at the destination node y, $p_{x \rightsquigarrow x}$ is a path that starts at node x and returns back to the same, and $p_{y \rightsquigarrow y}$ is a path that starts at node y and returns back to the same.

Example 8.6 Given the heterogeneous graph of Figure 8.7, we are asked to compute the *similarities* between the news articles by using simple *meta-paths*: ACA (content-based filtering), ASA (session-based filtering) and ASUSA (collaborative filtering). Table 8.4 shows the *similarities* of news article A3 with the other articles as follows: A3 shares only the same article category as A5, so $\text{PathSim}_{\text{ACA}}(\text{A3}, \text{A5}) = \frac{2*1}{1+1} = 1$. And A4 shares *sessions* with A3 and A5, therefore $\text{PathSim}_{\text{ASA}}(\text{A3}, \text{A4}) = \frac{2*1}{1+2} \approx 0.666$. Similarly, because articles A3 and A1 have been clicked (hence selected) by the same user, then $\text{PathSim}_{\text{ASUSA}}(\text{A3}, \text{A1}) = \frac{2*1}{1+1} = 1$. On the other hand, as articles A3 and A5 have not been clicked by the same user, then $\text{PathSim}_{\text{ASUSA}}(\text{A3}, \text{A5}) = \frac{2*0}{1+1} = 0$.

	ACA	ASA	ASUSA
similarity(A3, A1)	0	0	1
similarity(A3, A2)	0	0	1
similarity(A3, A4)	0	0.666	0.666
similarity(A3, A5)	1	0	0

Table 8.4: Calculated *similarity* between mews article A3 and the other articles using different *meta-paths* based on the PathSim algorithm.

Unlike the similarity measures *SimRank* and *RWR*, which are "biased" towards the most "well-

connected" nodes in the graph, *PathSim* takes into account the *"visibility"* or *"accessibility"* of nodes, bringing nodes that share similar *"accessibility"* closer together. For example, if two researchers have published 10 papers each and a third researcher has published 200 papers, *SimRank* and *RWR* would make the two aforementioned and comparatively inexperienced researchers very similar to the more experienced third, and thus less similar to each other. On the other hand, *PathSim* would reduce their *similarity* to the third and experienced researcher and bring them (the inexperienced ones) closer to each other, considering their small number of publications as another factor of their *similarity* (i.e., the inexperienced researchers, more similar to each other than to the experienced researchers).

8.4.6 Explanation of recommendations based on meta-paths

An important function of a recommendation algorithm, in addition to generating recommendations, is to be able to also justify them, so that the user can - in a "transparent" way - understand the reason that led to the recommendation of an item. In this direction, the PathSim algorithm is able to explain its recommendations based on the number of instances of a *meta-path* that led to a recommendation.

Example 8.7 In the graph of Figure 8.7, to provide recommendations to users based on similar news articles, we first find articles similar to the one the user just clicked on. Next, we rank them based on their *similarity* to the target news article that the user has interacted with by recommending a list of top-N news articles. Alternatively, for predicting the next article to be recommended, the user's entire last *session* can be taken into account to determine his/her *short-term* preferences in order to suggest the appropriate articles for the given time point. In this way, the *similarities* between articles can be determined by running *PathSim* on *meta-paths* ASA, AUA, ACA and ALA. For example, based on *meta-path* ASA, in Figure 8.10 we present how articles are ranked in a top-5 recommendation list, along with an explanation for each recommended news article.

Figure 8.10: News article recommendations based on *meta-path* ASA along with an appropriate explanation for each recommended article.

So as shown in Figure 8.10, we provide recommendations that reflect the overall user interest during a *session* along with appropriate explanations (i.e., why an article is recommended to the user). This is an intuitive and user-friendly way of explaining the relationship between the suggested article and the reason used to explain it, e.g. *these two news articles have been read together in 10 different user sessions.*

Next, we will present how we can combine several *meta-paths* to provide "hybrid" (multidimensional) explanations along with the recommendations. We will therefore use four *meta-paths* to provide an explanation for a recommended article: (AUA: when two articles have been read *by the same user*, ASA: when two articles have been read *within the same session*, ACA: when two articles *belong to the same article category*, and ALA: when two articles *refer to the same geographical location*).

Example 8.8 In our current example, combining the four *meta paths* [AUA, ASA, ACA, ALA], we get a hybrid 4-dimensional explanation, which is shown in Figure 8.11.

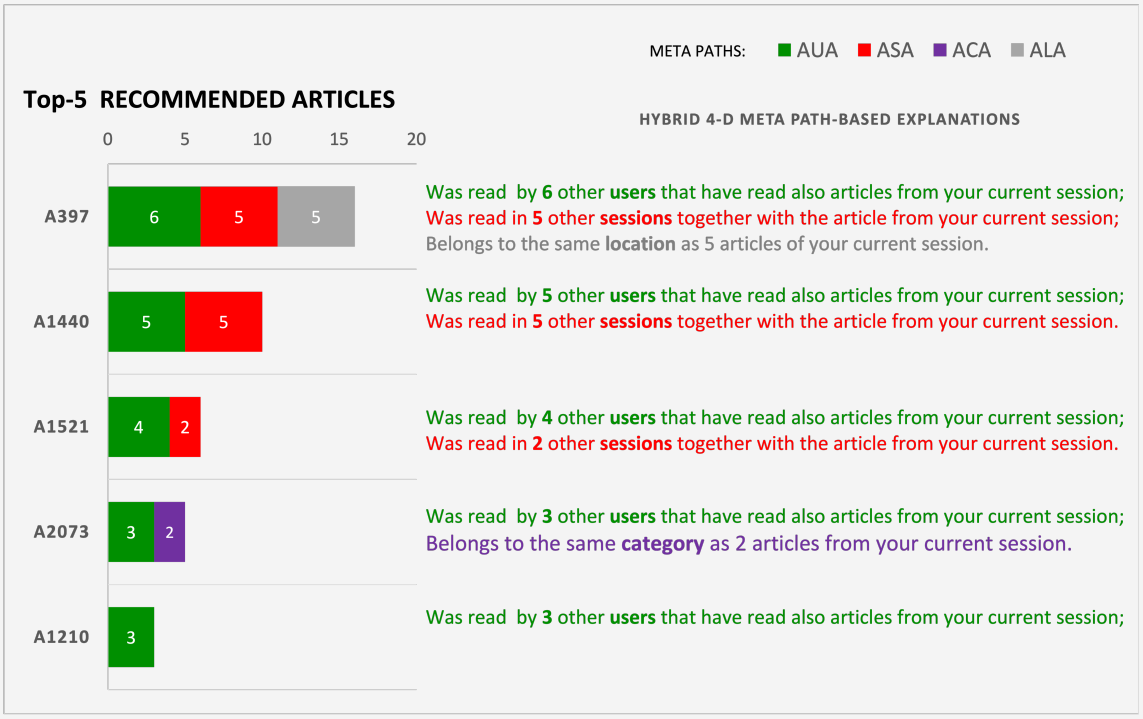


Figure 8.11: Hybrid 4-dimensional explanation based on *meta paths*.

As shown Figure 8.11, the top-5 recommended articles are ranked based on the total number

of *meta-paths* instances. Notice that we use different colors in the horizontal mixed bars for the different styles of explanations used (user, session, etc.), as well as the recommended news article.

8.5 Knowledge Graphs

The *RDF (Resource Description Framework)* is a *knowledge representation* language that allows the description of *resources* available in the real world. For our online newspaper example, such resources could be a journalist, an article, a reader, etc. The description of resources is done by defining relationships between them.

Statements or else known as *triplets* are the key element in *RDF* and are used to create assertions about a resource. RDF *triplets* can consist of the entity being described, the properties of the entity being described, and a value for each of its properties. Alternatively, the aforementioned RDF *triplets* may include three different things: a "subject", a "predicate", and an "object". It is emphasized that the "subject" and the "object" are *resources*, while the "predicate" is the *relationship* between them.

A *knowledge graph*, then, helps us to represent an RDF *triplet* with a *structure*, that contains *nodes* and *directed edges*. To both of them may be applied *labels*, which identify either the attributes of a node or the relationship connecting two nodes. The edges in an RDF *triplet* are always directed from "subject" to "object", while the *label of the edge* denotes the "predicate". In summary, the general structure of *triple RDF* tuples is as follows:

Statement (resource, property relationship, resource)

Example 8.9 Figure 8.12 shows the *Knowledge Graph* for an online newspaper that consists of journalists, news articles, article categories, regions and readers who interact with the news articles during their visit to the online newspaper. More specifically, journalists write about specific geographical areas (e.g. Thessaloniki, Athens, etc.). The geographical dimension has levels of granularity (city, county, etc.). The time dimension is also divided into different levels of granularity: Day, Week, Month, Year. Finally, readers explicitly state that they like specific articles, journalists and article categories. Suppose, therefore, that the owner of the online newspaper wants to develop on his behalf a *knowledge graph* that accumulates knowledge about the relationships between readers and articles, as well as other "entities" (journalists, article categories, etc.) with which they interact, in order to suggest personalized articles to readers based on their previous interaction with the articles and other entities in the knowledge graph.

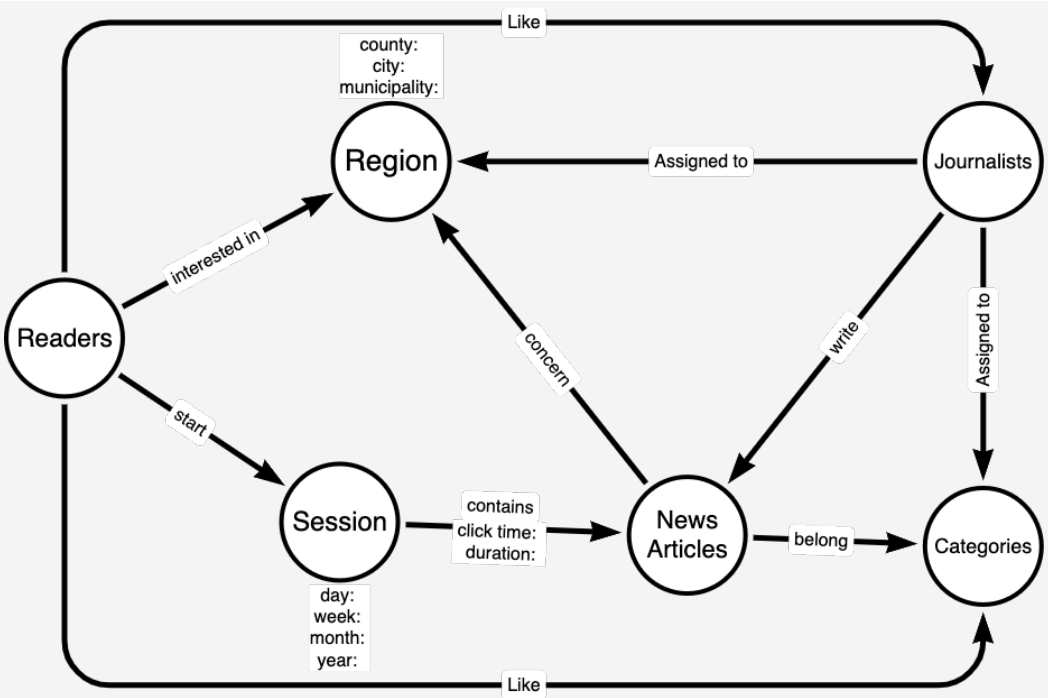


Figure 8.12: Example of an "ontology" (*RDF scheme*) related to an online newspaper

Therefore, based on the *knowledge graph* of Figure 8.12, we can recommend to the target user news articles of the same categories as those that s/he has interacted with in previous visits to the online newspaper. Suppose, therefore, that the target reader has interacted in the past with articles that belong to the "Politics" and "Sports" categories. In this case, the recommendation system based on the *knowledge graph* of Figure 8.12 finds the reader's previous *visits* (*sessions*) to the online newspaper and which articles he has read in them. It then retrieves the category to which these articles belong to, and suggests other articles belonging to the same categories, that the reader has not interacted with before.

A second recommendation example based on the *knowledge graph* of Figure 8.12 could be the following: we recommend to the reader, journalists who have written an article for a geographical area of interest to him/her. So suppose that the target reader has interacted with articles related to the geographical area of the city of Athens. If, now, we observe the hierarchy of the "entity" *Region* in Figure 8.12, we will see that it is broken down into 3 levels (Municipality, City, County). Therefore, as the recommendation system takes into account a higher level of abstraction (e.g. the county or else prefecture), it can conclude that the target reader could be interested in those newspaper columnists who are assigned to write for the prefecture of Attiki in Greece. Therefore, the recommendation system will suggest a similar list to the reader, possibly removing those articles s/he has already interacted with in the past.

8.6 Graph Convolutional Networks

One of the important studies in *GCN*-based recommendation systems is the research article, [Ying et al., 2018]. This paper describes the *PinSage algorithm*, which recommends items to users for a very large-scale online data service (meaning hundreds of millions of users and items), known as *Pinterest*. The proposed model is based on **random walks on the nodes** of the bipartite user-item graph. The aggregation of the latter creates latent representations of the nodes or else **node embeddings** of the graph in a smaller (more abstracted) vector space that is therefore more representative of each target node. By *node embedding* we mean the representation of a node in a smaller dimensional space, which is achieved by focusing on the main characteristics of the target node and removing any ”noisy” data. The training of the *GCN neural network* of the *PinSage algorithm* is guided by an **pair-wise ranking loss function**, so that relevant items are placed as close as possible to the newly created **embedding space**, and nodes are represented both in terms of their connectivity to other nodes and in terms of their attributes/features. To address the challenges of the high dimensional data, *Map Reduce* and *locality sensitive hashing* technologies are used since *Pinterest* is a service that manages hundreds of millions of users and items (usually photos) every day.

The problem of item recommendation in large-scale graphs is also addressed by [Chen et al., 2020]. Its authors revised the basic architecture of a *GCN* neural network by removing *nonlinear activation functions* and adding more links between *neurons of different layers*. Their motivation was the need to reduce the ”sparsity” of data in large user-item bipartite graphs and the fact that models based on the *GCN* architecture cannot easily grow in depth due to the **oversmoothing effect**, where node features tend to become more similar with the increase of the graph depth.

Finally, the problem of data ”sparsity” in user-item graphs is also addressed by [Feng et al., 2019]. The method proposed enriches the ratings of items from users with additional data sources (e.g., user demographics or other item attributes), both of which are embedded in the same vector space. They are then processed through *polynomial Chebyshev filters* to extract the basic user and item features. The authors also apply a user-based attention mechanism, so that *GCN* emphasizes to the most important neighbor users of the target user.

Next, we will describe the basic architecture of a *GCN* neural network with the help of an example.

Example 8.10 Suppose we are the administrators of an online publishing house that wants to provide personalized content to its readers based on their research preferences and their collaborations with other readers. Initially, the recommendation system may ask readers for some information about some broad demographic characteristics (e.g., occupation, research interests, etc.) that may play an important role in the selection of recommended articles. For example, a reader from an academic background tends to select scholarly articles related to

their individual research interests, while readers who work in research centres tend to select articles related to the subjects being studied there. Subsequently, as already mentioned, the recommendation system records readers' interaction with the articles as well as research or other collaborations between readers in order to profile them.

Based on the aforementioned scenario, we construct a user-item bipartite graph (see Figure 8.13) where the main characteristics of a reader are his/her profession and research interests. In the same Figure (8.13), the red directed edges connecting readers to articles capture the fact that a reader likes a particular research article. Finally, in Figure 8.13, the green edges capture the aforementioned collaborations of each reader with others. For a new reader, for whom there are not many interactions with articles yet, there is the so-called **cold-start** problem, since the recommender system needs a minimum number of interactions of the reader with the articles in order for its recommendations to be accurate and precise. In such a difficult case, the recommendation system could represent a reader's profile by using the profiles of other users or articles that have similar *node features* with him/her.

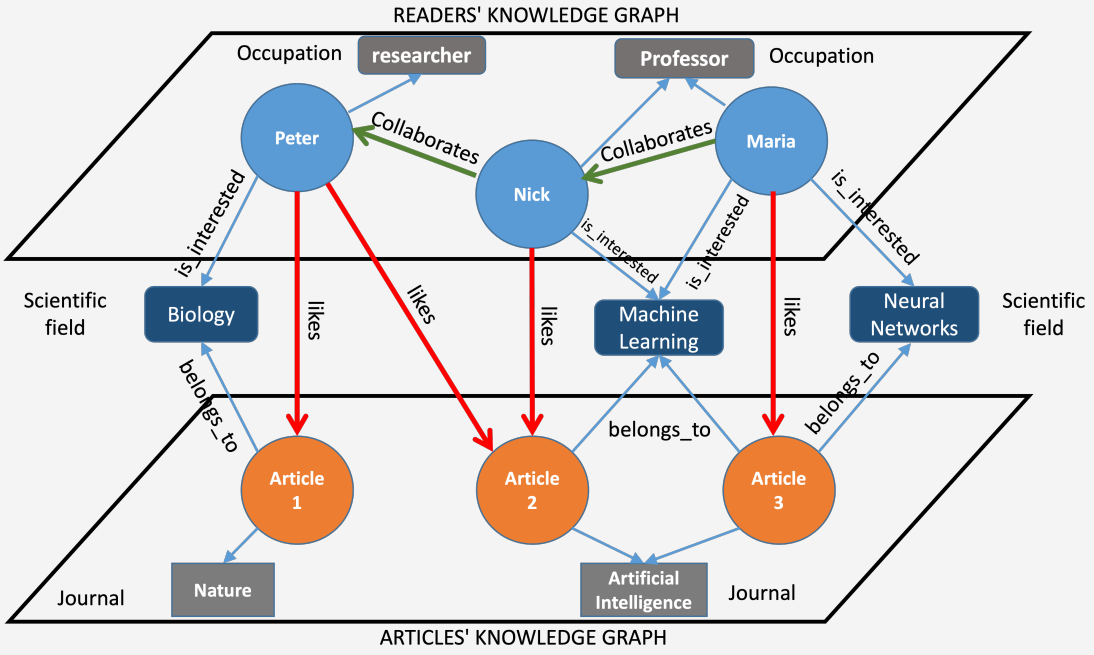


Figure 8.13: Readers-Articles Knowledge Graph.

Suppose we are asked to predict the characteristics (occupation and research interests) of the reader named Peter in Figure 8.13, based on the articles he likes and his collaborations with other readers. Therefore, we form a new graph in order to predict the characteristics of reader Peter, who is located at the center of the *concentric circles/convolution layers* of a GCN neural network as shown in Figure 8.14. The graph consists of two types of nodes (readers and articles). As we observe in Figure 8.14, the two features (profession and research interests) of each

reader and the two features (journal title and scientific domain) of each article appear as **node features** of the graph. As it is also shown in Figure 8.14, the preferences of Peter in terms of scientific articles and his collaborations with other researchers are represented by using directed edges. In particular, there are two different types of edges here (see Figure 8.14): The first type (edges in green) refers to Peter's collaborations, while the second type (edges in red) refers to his preferences for specific articles. Notice that all edges are directed towards the target node (Peter), because for this node we wish to compute its latent representation (i.e., its node embedding) in the newly constructed smaller latent space.

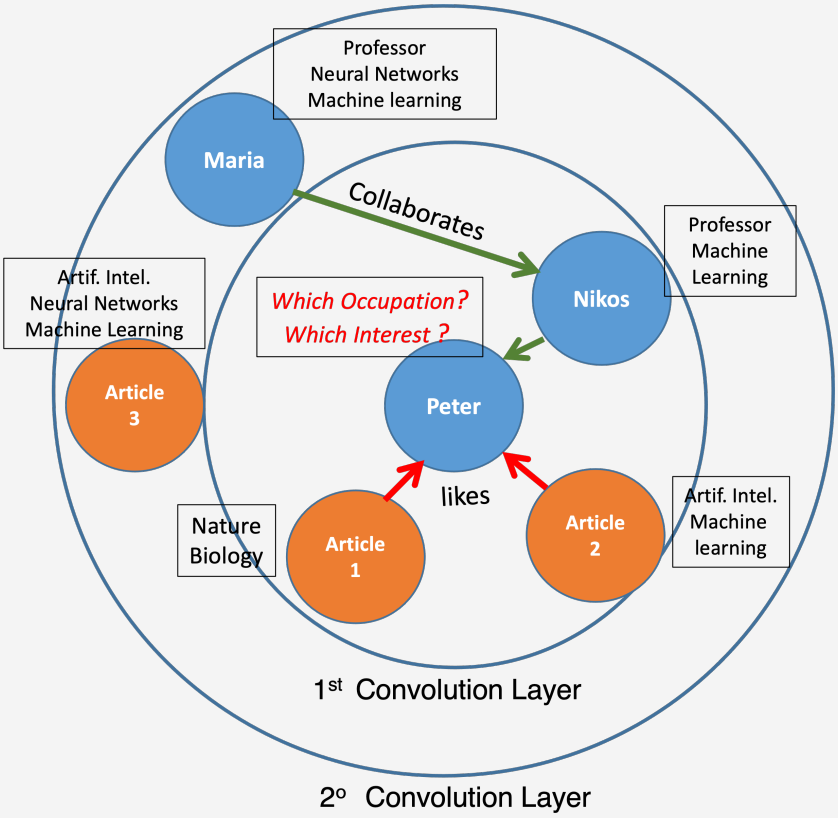


Figure 8.14: Concentric convolution layers of the GCN neural network for generating the latent representation of the node "Peter" based on the data of the Example 8.10.

The basic function of a *Graph Convolutional Network (GCN)* is to gather information from the adjacent nodes of the target node and forward it to the next convolution layer. Therefore, to predict the latent vector of the reader named Peter, we build a *GCN* neural network based on the data of the graph in Figure 8.14 such that the characteristics (research interests, occupation, etc.) of the neighboring nodes of the target node are propagated and aggregated cumulatively from layer to layer to represent its own preferences based on the following convolution rule:

$$h_i^{(l+1)} = \sigma\left(\sum_{r\in R}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,r}}\cdot W_r^{(l)}\cdot h_j^{(l)}\right) \tag{8.13}$$

where $h_i^{(l+1)}$ is the representation of the node's feature of the target node i, $h_j^{(l)}$ is the latent representation of the neighbouring node j in the l-layer of the convolutionary neural network, $W_r^{(l)}$ the weight matrix in the l-layer with respect to the directed edges of type r, $\mathcal{N}_i^r$ is the set of neighbors of the target node i that are connected to it by the directed edges of type $r \in \mathcal{R}$, and $c_{i,r} = |\mathcal{N}_i^r|$ is the number of neighbors of node i with respect to the directed edges of type r. Recall that there are two types of directed edges r in our example: the edges of collaborations with other readers and those indicating Peter's preference for an article. Finally, σ is a *nonlinear activation function* (e.g., the *sigmoid function*).

In summary, the representation of Peter's preferences via a *GCN* neural network can be derived from the cumulative aggregation of the information of his neighboring nodes based on either (a) his interactions with articles or (b) his collaborations with other readers. For example, the latent vector of the target reader i based on its interaction with articles is as follows:

$$\mathrm{h}_i = \sigma\left(\mathbf{W}\cdot \mathrm{AGGREGATE}\left(\left\{\mathrm{q}_j, \forall j \in C(i)\right\}\right)\right) \tag{8.14}$$

where $C(i)$ is the set of articles liked by reader i, and q_j is the *latent* vector of an article j with which reader i has interacted. The AGGREGATE is a *generalized function of cumulative aggregation of the features of the neighboring nodes* (articles or readers) of the target node i. Usually for the *AGGREGATE* function we use the *averaging* operator, which is shown in the following Equation:

$$\mathrm{h}_i = \sigma\left(\mathbf{W}\cdot\left\{\sum_{j\in C(i)}\frac{1}{C(i)}\cdot \mathrm{q}_j\right\}\right) \tag{8.15}$$

Alternatively, instead of the *average* operator we could use the *concatenation* operator, which simply concatenates the vectors of the nodes that are aggregated together, as shown below:

$$\mathrm{h}_i = \sigma\left(\mathbf{W}\cdot\left\{\sum_{j\in C(i)}\left[\,\mathrm{h}_i \parallel \mathrm{q}_j\right]\right\}\right) \tag{8.16}$$

where $\parallel$ is the operator of the concatenation of the two vectors expressing the characteristics of the nodes being concatenated.

Based on the data in Figure 8.14, let us assume that we will use the *aggregation operator* CONCATENATION. Therefore, given the characteristics of nodes *Article 1* and *Article 2*, we would make the prediction about Peter's research interests that they would be either in *Biology* or *Machine Learning*. We emphasize that the above cumulative aggregation procedure should

also be followed for Peter's collaborations with other readers who both work in academia. Therefore, based on his aforementioned collaborations, we expect Peter's occupation to be University Professor (see Figure 8.14).

To summarize, in this section we considered how we can represent a node of a graph in a *low-dimensional vector representation* (*node embedding*). In particular, we predicted the vector representation of the reader named Peter as shown in Figure 8.14.

8.6.1 Graph Attention Mechanism

In this section we will redefine the propagation rule of a *GCN*, which is based on Equation 8.13 by applying an *attention mechanism*, so that through this fixed coefficient -here $c_{i,r}$- which represents the number of neighbors of the target node i for edges of type r, we will not assign the same importance to all neighboring nodes.

The basic idea behind the **Graph Attention Mechanism** is to dynamically compute for each node the coefficient $c_{i,r}$ of Equation 8.13 rather than simply compute a fixed value of $c_{i,r}$ for all nodes. Thus, we can use the characteristics of the nodes to determine the "weight" of the most important nodes with respect to the target node. Therefore, some neighboring nodes may be more important than others in determining the representation of the target node because they have more relevant *node features* with it.

The new dynamic coefficient will henceforth be denoted as a_{ij}, and will be calculated based on the common features of a neighboring node j with the target node i, which will then be passed to a *attention function* implemented by a *single-layer perceptron* as follows:

$$a_{ij} = \text{attention}\left(h_i, h_j\right)$$

Now, for a target node i, and in order to make the *attention coefficients* a_{ij} easily comparable among its neighbors, we normalize them with respect to the sum of the coefficients of all neighboring nodes j using the *softmax* function, so as to obtain a probability distribution whose sum is unity (1), as follows:

$$a_{ij} = \frac{\exp\left(a_{ij}\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(a_{ik}\right)}$$

Thus, the *convolution rule and propagation* of Equation 8.13 is reformulated as follows:

$$h_i^{(l+1)} = \sigma\left(\sum_{r \in R} \sum_{j \in \mathcal{N}_i^r} a_{ij} \cdot W_r^{(l)} \cdot h_j^{(l)}\right) \tag{8.17}$$

$$h_i^{(l)} = \sigma\left(\sum_{i \in N_j} a_{ij} \cdot W \cdot h_j\right)$$

8.6.2 Node representation in GCNs by using Linear Algebra Matrices

In this subsection we will define the *convolution rule and the information propagation* of Equation 8.13 by using linear algebra matrices. So based on the **adjacency matrix A** $\in \{1, 0\}^{n \times n}$ of nodes and the **feature matrix** of nodes $\mathbf{H} \in \mathbb{R}^{n \times c}$ we can express the *information convolution rule* of Equation 8.13 in the form of a product of matrices as follows:

$$\mathrm{H}^{(l+1)} = \sigma\left(\tilde{\mathrm{D}}^{-1} \cdot \tilde{\mathrm{A}} \cdot \mathrm{H}^{(l)} \cdot \mathrm{W}^{(l)}\right)$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathrm{I}$, I is the *identity matrix*, $\tilde{D}$ is a diagonal **degree matrix** with $\tilde{D}_{i,i} = \sum_j \tilde{\mathbf{A}}_{i,j}$, $\mathbf{W} \in \mathbb{R}^{c \times c'}$ a weight matrix, σ a *nonlinear activation function* and $\mathbf{H} \in \mathbb{R}^{n \times c^{prime}}$ the *latent representation of the nodes' features of the graph.* The rationale behind the above equation is that the initial representations **H** of the nodes are subject to a linear transformation through their multiplication by the weight matrix **W**, and are then propagated to neighboring nodes via the *transition probability matrix* $\tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$.

In addition, the **spectral decomposition** of the *Laplacian matrix* **L** allows the construction of low-dimensional node embeddings in graphs. The *Laplacian matrix* **L** is a representation of a graph in a matrix form, as illustrated below:

$$\mathrm{L} = \mathrm{I} - \mathrm{D}^{-\frac{1}{2}} \cdot \mathrm{A} \cdot \mathrm{D}^{-\frac{1}{2}}$$

We can also use an alternative definition of *Laplacian matrix* as follows:

$$\mathrm{L} = \mathrm{D}^{-\frac{1}{2}} \cdot (\mathrm{D} - \mathrm{A}) \cdot \mathrm{D}^{-\frac{1}{2}}$$

The above matrix **L** is known as *signed normalized Laplacian matrix with positive/negative sign.* Therefore, we can express the *convolution rule and the information propagation* of Equation 8.13 in the form of a matrix product as shown below:

$$\mathrm{H}^{(l+1)} = \sigma\left(\mathrm{L} \cdot \mathrm{H}^{(l)} \cdot \mathrm{W}^{(l)}\right)$$

where L is the *Laplacian matrix*, H is the matrix that holds the features of the graph nodes, and W is the weight matrix.

8.6.3 Loss Function Definition for Predicting Ratings in GCNs

For the purposes of a recommender system, the output of the neural network usually should be the prediction of a user's rating over an item. As we explained in Section 6.1.2, by using an *MLP* it is possible to consider the target user and an item as two independent inputs and predict a rating by taking their *dot product.* When we use a GCN, we can insert the convolved vectors into a three-layer

MLP to learn the potential correlation between the user and the item embeddings, so that, we can predict a rating for them, as follows:

$$\mathrm{z}_1 = \mathrm{h}_i \oplus \mathrm{q}_j \tag{8.18}$$

$$\mathrm{z}_l = \sigma\left(W_l^T \mathrm{z}_{l-1} + b_l\right) \tag{8.19}$$

$$\hat{y}_{ij} = \sigma\left(W^T \mathrm{z}_l + b\right) \tag{8.20}$$

where h_i is the embedding of the user, q_j is the embedding of the item, $\oplus$ is the *concatenation*, and the variables W_l, b_l and σ denote the weight matrix, the *bias* and the *activation function* for the l-layer respectively. Finally, we define the *cross-entropy loss function* of the predicted rating $\hat{y}_{ij}$ as follows:

$$L = -\sum_{(i,j)\in Y\cup Y^-} y_{ij} \cdot \log \hat{y}_{ij} + \left(1 - y_{ij}\right) \cdot \log\left(1 - \hat{y}_{ij}\right) \tag{8.21}$$

where $\hat{y}_{ij}$ is the rating we predict that the target user i would give to the item j, while y_{ij} is the actual rating he has already given. We emphasize here that the augmented dataset Y^- includes, in addition to the observed user-item interaction pairs Y, also the user-item interaction pairs – that have not actually existed (unobserved pairs), so that we have a *balance* in terms of training the prediction model between positive (observed) and negative (unobserved) user-item interactions.

8.7 Graph Embeddings

Graph embeddings are sophisticated methods of reducing the dimensionality of the node features of a graph, which have led to more efficient **graph data analytics**. For this reason, graph embeddings have been used in recommender systems for **similarity inference** and **link prediction** between users and items of a bipartite graph. For example, the *node2vec* and *DeepWalk* [Grover and Leskovec, 2016] algorithms learn vector representations of the nodes of a graph to identify semantically "similar" nodes. In particular, they learn latent representations of the nodes of a graph by "traversing" the graph using *random walks*. The two aforementioned algorithms [Grover and Leskovec, 2016] essentially apply the *Skip-gram* model to learn the latent representations of the nodes of a graph (instead of the word representations of a given text, where the aforementioned model, known in its above version as *word2vec*, was originally applied). In the same direction, the *Metapath2vec* algorithm [Dong et al., 2017] extends the *DeepWalk* and *node2vec* algorithms as it is applied to heterogeneous graphs by performing *random walks* based on **meta-paths**.

In addition, there are architectures of *deep neural networks*, such as *graph neural networks* (*GNNs*), *graph convolution networks* (*GCNs*), etc, which propagate and cumulatively aggregate iteratively both the *node features* and the *node locality structure* through the local neighbourhoods of the target node.

Their purpose is to "learn" a latent vector of the aforementioned node that is more representative compared to the original features vector. The *MCRec* method [Hu et al., 2018], for example, is an extension of *Neural Collaborative Filtering (NCF)* [He et al., 2017] method and uses as input the node embeddings constructed from *node2vec* algorithm, which is a variation of the *word2vec* algorithm for graphs. Finally, the *MP4Rec* [Ozsoy et al., 2020] method is a *GNN* which is able to provide both accurate and explainable recommendations. To learn the local characteristics of the nodes of the graph, it uses *meta-paths* and the *PathSim* algorithm [Yizhou et al., 2011].

8.8 Chapter Questions

1. List the local-based similarity measures between nodes of a graph. For each, define how to compute it.
2. Describe the *Katz Status index algorithm* for measuring *similarity* between nodes of a graph.
3. Describe briefly the *SimRank* algorithm for measuring *similarity* between nodes of a graph.
4. Describe the *Random Walk with Restart* algorithm for measuring *similarity* between nodes of a graph.
5. Describe the PathSim algorithm for measuring *similarity* between nodes of a graph.
6. Apply to the graph in Figure 8.15 the following simple *PageRank* without "teleportation" Equation:

$$\mathrm{v}' = \mathrm{M} \cdot \mathrm{v}$$

where M is the transition probability matrix from one node to another and v is a vector expressing the importance of each node in the graph based on its accessibility/visibility. Please recall that the idea behind *PageRank* is based on the fact that a random "walker" starts at a random node and each time randomly transitions to an adjacent node until it completes its "journey". At the end of this "journey" the task is to find out which nodes were visited most often, which means that these are the most similar to the target node.

   (a) Initialize each value of the initial probability vector as follows: $\mathrm{v}_0 = \mathrm{e}/n$, where n is the number of nodes in the graph, and e is a vector of length n with all elements equal to 1 (unity). What will be the values of the initial vector v_0?
   (b) Calculate the *transition probability matrix* M from one node to the others.
   (c) Apply the simple formula of *PageRank* above and run it iteratively for three steps or until the vector v' converges (1e-4 convergence factor). What will be the final values of PageRank similarity vector v'?

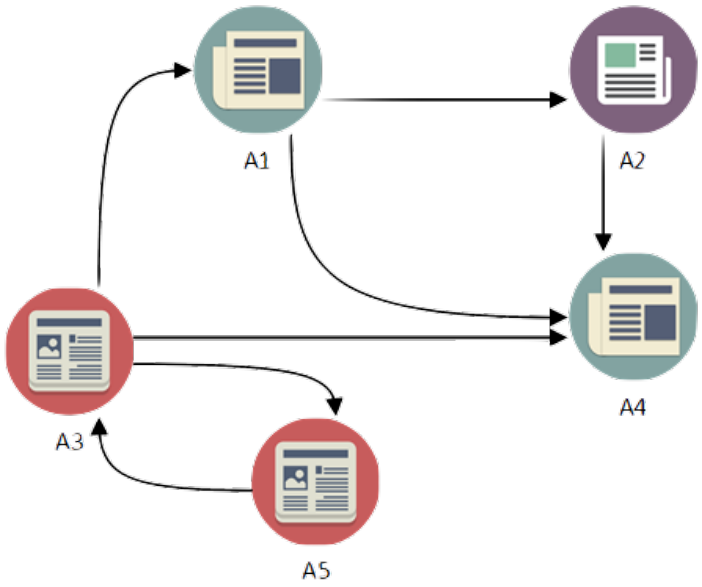


Figure 8.15: A *graph* of news articles, which cite/refer each other.

(d) What is the equation for computing the *node embedding* in a *Graph Convolutional Network*? Describe each parameter of the equation separately.

(e) Describe the *Graph Attention Mechanism*. What is the basic operation it performs in a GCN neural network?

7. Apply the *PageRank* with "teleportation" algorithm to the graph of Figure 8.15, where at each step of the algorithm the "walker" will follow a "random path" with probability β, and with probability $(1-\beta)$ it can jump to a random node of the graph, using the following equation:

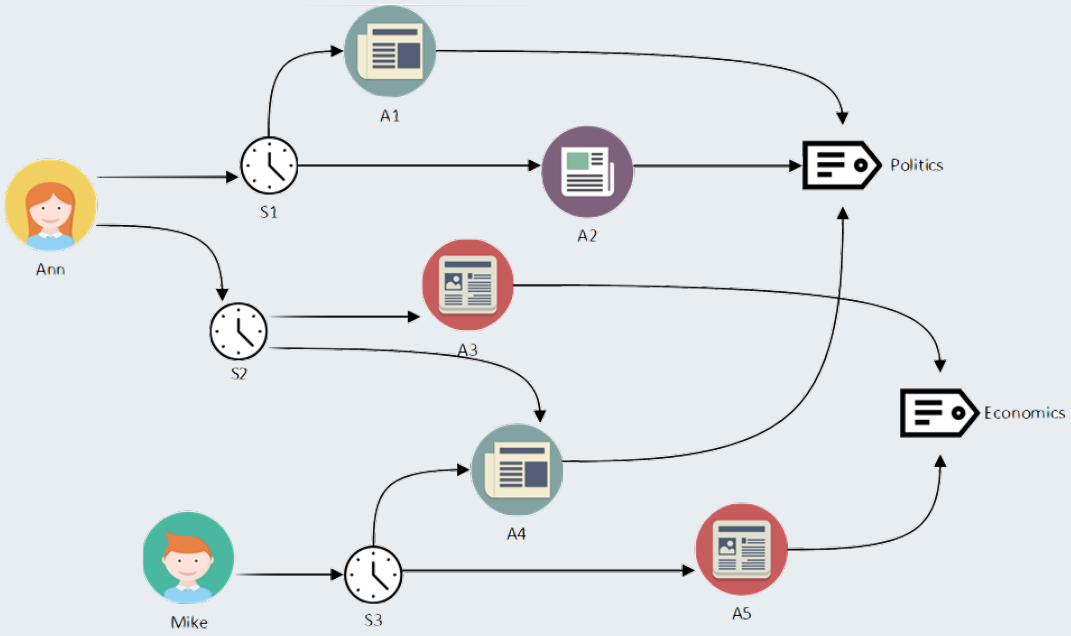
$$\mathrm{v}' = \beta \times \mathrm{M} \times \mathrm{v} + (1-\beta) \times \mathrm{e}/n$$

where n is the number of nodes in the network and e is a vector of length n with all elements equal to 1 (unity). Apply the formula of *PageRank* with "teleportation" for three iterations. Which result do you get? Use $\beta = 0.8$.

8.9 Exercises in Programming

8.9.1 Heterogeneous Information Network for an Online Newspaper

We are given the following graph consisting of 4 different types of nodes (users, sessions, articles, article categories):



Requirements:

1. Create a *heterogeneous graph* with the four different types of nodes and present it visually.
2. Run the *RWR algorithm* (*random walk with restart*) on the entire *heterogeneous network*.
3. Analyze the AA (*Article-Article*), UA (*User-Article*) and UC (*User-Category*) subtables by answering the following questions:
   (a) For subtable AA:
   - What is the most *relevant* article to the target article A1? Why?
   - Which article is more similar to target article A2: A3 or A4? Why?

   (b) For subtable UA:
   - If we were to recommend to Ann again an article that she may have already read, what would it be? Also, what article would we recommend to Mike again?
   - Why are articles A1 and A2 more likely to be re-read by Ann than article A3?

   (c) For subtable UC:
   - Why is the Economics category more interesting to Mike than the Politics category?

Solutions:

1. In the following, we give the result relevance matrix after running the *random walk with restart* algorithm:

	Ann	Mike	S1	S2	S3	A1	A2	A3	A4	A5	Politics	Economics
Ann	0.2841	0.0089	0.1625	0.1525	0.0332	0.0612	0.0612	0.0524	0.0674	0.0206	0.0669	0.0291
Mike	0.0178	0.2764	0.0196	0.0469	0.2865	0.0157	0.0157	0.0335	0.0994	0.0974	0.0389	0.0523
S1	0.1083	0.0065	0.3393	0.0672	0.0245	0.1198	0.1198	0.0236	0.0538	0.0123	0.1103	0.0144
S2	0.1016	0.0156	0.0672	0.3141	0.0588	0.0331	0.0331	0.1071	0.1146	0.0390	0.0571	0.0585
S3	0.0222	0.0955	0.0245	0.0588	0.3583	0.0195	0.0195	0.0418	0.1241	0.1216	0.0488	0.0654
A1	0.0612	0.0078	0.1797	0.0497	0.0292	0.2942	0.0942	0.0182	0.0673	0.0128	0.1732	0.0124
A2	0.0612	0.0078	0.1797	0.0497	0.0292	0.0942	0.2942	0.0182	0.0673	0.0128	0.1732	0.0124
A3	0.0524	0.0167	0.0355	0.1607	0.0627	0.0182	0.0182	0.3040	0.0684	0.0778	0.0327	0.1527
A4	0.0450	0.0331	0.0538	0.1146	0.1241	0.0449	0.0449	0.0456	0.2942	0.0481	0.1143	0.0374
A5	0.0206	0.0487	0.0184	0.0585	0.1825	0.0128	0.0128	0.0778	0.0722	0.3109	0.0294	0.1554
Politics	0.0446	0.0130	0.1103	0.0571	0.0488	0.1155	0.1155	0.0218	0.1143	0.0196	0.3229	0.0166
Economics	0.0291	0.0261	0.0216	0.0878	0.0981	0.0124	0.0124	0.1527	0.0561	0.1554	0.0249	0.3233

2. In the following, there is a detailed commentary of the results for the AA, UA and UC subtables:

   (a) For subtable AA:

   - Article A2 is the most relevant to A1, because it has been clicked on by the same user in the same session and they belong to the same category of articles.
   - A4 is most similar to A2 because it has been clicked on by the same user and belongs to the same article category, while A3 and A2 have only been clicked on by the same user.

   (b) For subtable UA:

   - We would suggest that both Ann and Mike re-read article A4, because it is the most popular article on the graph.
   - We can observe that Ann shows more interest in political articles than in economics (3 out of the 4 articles she has read belong to Politics). That is why the *strength* of A1 and A2 is higher than that of A3.

   (c) For subtable UC:

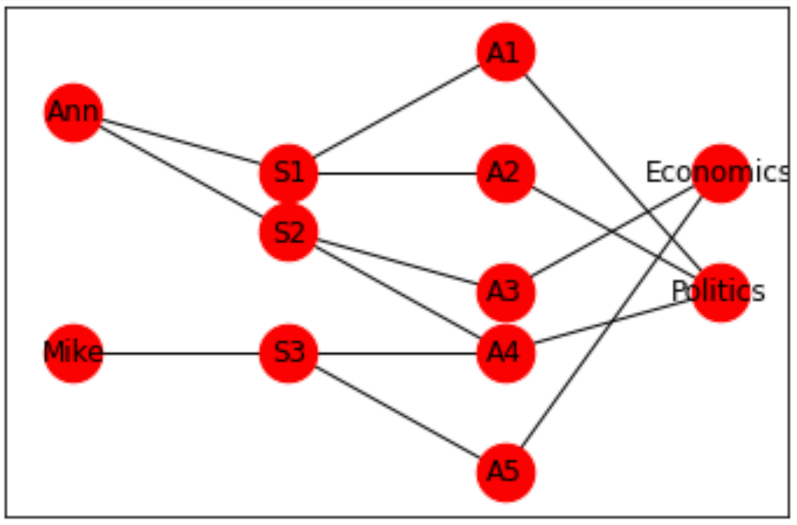
   - Although Mike has read a Politics article and an Economics article, we can observe that their scores are not the same because the Politics article A4 - as just mentioned above - appeared in the same session as the Economics article A3 read by Ann, which brings the two categories closer to each other. Hence, A4 article comes a little closer to Economics, which in turn means that Mike prefers Economics articles more than Politics ones.

In the following, we present the *Python* code, which implements the exercise in programming:

```
#Import Libraries
import networkx as nx
```

```
import pandas as pd
import numpy as np
from collections import defaultdict
import copy

# Heterogeneous Graph Creation
G = nx.Graph() # undirected  graph
G.add_nodes_from(['Ann','Mike'], entity='U')
G.add_nodes_from(['S1','S2','S3'], entity='S')
G.add_nodes_from(['A1','A2','A3','A4','A5'], entity='A')
G.add_nodes_from(['Politics','Economics'], entity='C')

G.add_edges_from([('Ann','S1'), ('Ann','S2'), ('Mike','S3')])

G.add_edges_from([('S1','A1'), ('S1','A2'),
                  ('S2','A3'), ('S2','A4'),
                  ('S3','A4'), ('S3','A5')])

G.add_edges_from([('A1','Politics'), ('A2','Politics'), ('A4','
    Politics'), ('A3','Economics'), ('A5','Economics')])

positions = {'Ann':[0,5],'Mike':[0,1],
             'S1':[2,4],'S2':[2,3],'S3':[2,1],
             'A1':[4,6],'A2':[4,4],'A3':[4,2],'A4':[4,1],'A5'
    :[4,-1],
             'Politics':[6,2],'Economics':[6,4]}

nx.draw_networkx(G,node_size=600, with_labels=True, node_color=
    'red',pos=positions)
```

In the following the visual representation of our graph:




```
# Adjacency Matrix A Creation
A = nx.adjacency_matrix(G)
A.todense()
matrix([[0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0],
        [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0],
        [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0],
        [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1],
        [0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0],
        [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1],
        [0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0]])

# Creation of the Transition Probability Matrix
A.todense()
# Calculation of the Nodes' Degree Matrix
# d = np.array(list(G.degree().values()))
d = np.array([len(nbrs) for n,nbrs in G.adj.items()])


p = 1./d
```

```
25
26 M = A.multiply(p)
27 d
28 #array([2, 1, 3, 3, 3, 2, 2, 2, 3, 2, 3, 2])
29
30 #parameters initialization of the RWR algorithm
31 beta = 0.8
32 max_iter = 100
33 eps = 1e-4
34
35 n = nx.number_of_nodes(G)
36 I = np.identity(n)
37
38 # RWR algorithm implementation
39 V = I
40 for i in range(max_iter):
41     V_new = beta*M*V + (1-beta)*I
42
43     if (abs(V-V_new) < eps).all():
44         print("Converged after %d iteration" % (i))
45         break
46
47     V = V_new
48 V.round(4)
49
50 #Canverges after  34 iterations
51 array([[0.2838, 0.0178, 0.1083, 0.1016, 0.0222, 0.0612, 0.0612,
       0.0524, 0.045 , 0.0206, 0.0446, 0.0291],
52        [0.0089, 0.2764, 0.0065, 0.0156, 0.0955, 0.0078, 0.0078,
       0.0167, 0.0331, 0.0487, 0.013 , 0.0261],
53        [0.1625, 0.0196, 0.3393, 0.0672, 0.0245, 0.1797, 0.1797,
       0.0355, 0.0538, 0.0184, 0.1103, 0.0216],
54        [0.1525, 0.0469, 0.0672, 0.3141, 0.0588, 0.0497, 0.0497,
       0.1607, 0.1146, 0.0585, 0.0571, 0.0878],
55        [0.0332, 0.2865, 0.0245, 0.0588, 0.3583, 0.0292, 0.0292,
       0.0627,0.1241, 0.1825, 0.0488, 0.0981],
```

```
       [0.0612, 0.0157, 0.1198, 0.0331, 0.0195, 0.2942, 0.0942,
    0.0182, 0.0449, 0.0128, 0.1155, 0.0124],
       [0.0612, 0.0157, 0.1198, 0.0331, 0.0195, 0.0942, 0.2942,
    0.0182, 0.0449, 0.0128, 0.1155, 0.0124],
       [0.0524, 0.0335, 0.0236, 0.1071, 0.0418, 0.0182, 0.0182,
    0.304 , 0.0456, 0.0778, 0.0218, 0.1527],
       [0.0674, 0.0994, 0.0538, 0.1146, 0.1241, 0.0673, 0.0673,
    0.0684, 0.2942, 0.0722, 0.1143, 0.0561],
       [0.0206, 0.0974, 0.0123, 0.039 , 0.1216, 0.0128, 0.0128,
    0.0778, 0.0481, 0.3109, 0.0196, 0.1554],
       [0.0669, 0.0389, 0.1103, 0.0571, 0.0488, 0.1732, 0.1732,
    0.0327, 0.1143, 0.0294, 0.3229, 0.0249],
       [0.0291, 0.0523, 0.0144, 0.0585, 0.0654, 0.0124, 0.0124,
    0.1527, 0.0374, 0.1554, 0.0166, 0.3233]])
```

Note: For easier and better analysis of the above result table we can convert it to a *data frame* format using the following *Python* code:

```

result = defaultdict(list)
for n1 in G.nodes():
  result[n1] = defaultdict(int)
  for n2 in G.nodes():
      result[n1][n2] = V[n_dict[n2]][n_dict[n1]]

order = ['Ann','Mike','S1','S2','S3','A1','A2','A3','A4','A5','
    Politics','Economics']
result_df = result_df[order]
result_df.reindex(order)
```

	Ann	Mike	S1	S2	S3	A1	A2	A3	A4	A5	Politics	Economics
Ann	0.2841	0.0089	0.1625	0.1525	0.0332	0.0612	0.0612	0.0524	0.0674	0.0206	0.0669	0.0291
Mike	0.0178	0.2764	0.0196	0.0469	0.2865	0.0157	0.0157	0.0335	0.0994	0.0974	0.0389	0.0523
S1	0.1083	0.0065	0.3393	0.0672	0.0245	0.1198	0.1198	0.0236	0.0538	0.0123	0.1103	0.0144
S2	0.1016	0.0156	0.0672	0.3141	0.0588	0.0331	0.0331	0.1071	0.1146	0.0390	0.0571	0.0585
S3	0.0222	0.0955	0.0245	0.0588	0.3583	0.0195	0.0195	0.0418	0.1241	0.1216	0.0488	0.0654
A1	0.0612	0.0078	0.1797	0.0497	0.0292	0.2942	0.0942	0.0182	0.0673	0.0128	0.1732	0.0124
A2	0.0612	0.0078	0.1797	0.0497	0.0292	0.0942	0.2942	0.0182	0.0673	0.0128	0.1732	0.0124
A3	0.0524	0.0167	0.0355	0.1607	0.0627	0.0182	0.0182	0.3040	0.0684	0.0778	0.0327	0.1527
A4	0.0450	0.0331	0.0538	0.1146	0.1241	0.0449	0.0449	0.0456	0.2942	0.0481	0.1143	0.0374
A5	0.0206	0.0487	0.0184	0.0585	0.1825	0.0128	0.0128	0.0778	0.0722	0.3109	0.0294	0.1554
Politics	0.0446	0.0130	0.1103	0.0571	0.0488	0.1155	0.1155	0.0218	0.1143	0.0196	0.3229	0.0166
Economics	0.0291	0.0261	0.0216	0.0878	0.0981	0.0124	0.0124	0.1527	0.0561	0.1554	0.0249	0.3233

Bibliography

L. Adamic and E. Adar. How to search a social network. *Social Networks*, 27(3):187–203, 2005.

A. L. Barabasi, H. Jeong, Z. Neda, E. Ravasz, A. Schubert, and T. Vicsek. Evolution of the social network of scientific collaborations. *Physica A*, 311:590–614, 2002.

J. Chen, W. Geyer, C. Dugan, M. Muller, and I. Guy. Make new friends, but keep the old: recommending people on social networking sites. In *Proceedings of the 27th International Conference on Human Factors in Computing Systems (CHI)*, pages 201–210, Boston, MA, 2009.

L. Chen, L. Wu, R. Hong, K. Zhang, and M. Wang. Revisiting graph based collaborative filtering: A linear residual graph convolutional network approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 27–34, 2020.

G. G. Chowdhury. *Introduction to modern information retrieval.* Facet publishing, 2010.

Y. Dong, V. Chawla, and A. Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 135–144. ACM, 2017.

C. Feng, Z. Liu, S. Lin, and T. Quek. Attention-based graph convolutional network for recommendation system. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7560–7564. IEEE, 2019.

K. C. Foster and S. Q. Muth. A Faster Katz Status Score Algorithm. *Computational & Mathematical Organization Theory*, 7(4):275–285, 2002.

M. Fredman and R. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596–615, 1987.

A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

I. Guy, I. Ronen, and E. Wilcox. Do you know?: recommending people to invite into your social network. In *Proceedings of the 13th International Conference on Intelligent User Interfaces (IUI)*, pages 77–86, Sanibel Island, FL, 2009.

T. Haveliwala. Topic sensitive page rank. In *World Wide Web Conference(WWW)*, Honolulu, Hawaii, USA, 2002.

X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, pages 173–182. International World Wide Web Conferences Steering Committee, 2017.

B. Hu, C. Shi, W. X. Zhao, and P. S. Yu. Leveraging meta-path based context for top-n recommendation with a neural co-attention model. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1531–1540. ACM, 2018.

G. Jeh and J. Widom. Simrank: a measure of structural-context similarity. In *Proceedings 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'2002)*, pages 538–543, Edmonton, Canada, 2002.

L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, Mar. 1953. ISSN 0033-3123. 10.1007/BF02289026.

J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of massive data sets*. Cambridge university press, 2019.

D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *Proceedings of the 12th International Conference on Information and Knowledge Management (CIKM)*, pages 556–559, New Orleans, LO, 2003.

M. E. J. Newman. Clustering and preferential attachment in growing networks. *Physical Review E*, 64(2), 2001.

M. G. Ozsoy, D. O'Reilly-Morgan, P. Symeonidis, E. Z. Tragos, N. Hurley, B. Smyth, and A. Lawlor. Mp4rec: Explainable and accurate top-n recommendations in heterogeneous information networks. *IEEE Access*, 8:181835–181847, 2020.

L. Page and S. Brin. The pagerank citation ranking: Bringing order to the web. In *World Wide Web Conference(WWW)*, 1998.

J. Pan, H. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 653–658, Seattle, WA, 2004.

A. Papadimitriou, P. Symeonidis, and Y. Manolopoulos. Friendlink: Link prediction in social networks via bounded local path traversal. In *Proceedings of the 3rd Conference on Computational Aspects of Social Networks (CASON)*, pages 66–71, Salamanca, Spain, 2011.

Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment*, 4(11):992–1003, 2011.

H. Tong, C. Faloutsos, and J. Pan. Fast random walk with restart and its applications. In *ICDM '06: Proceedings of the 6th International Conference on Data Mining*, pages 613–622. IEEE Computer Society, 2006. ISBN 0-7695-2701-9. http://dx.doi.org/10.1109/ICDM.2006.70.

R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.

S. Yizhou, H. Jiawei, Y. Xifeng, Y. Philip S., and W. Tianyi. Pathsim: Meta path-based top-k similarity search in heterogenuos information networks. In *Proceedings of the VLDB Endowment (VLDB'2011)*, Seattle, Washigton, 2011.