Chapter 2
Representations of Networks

2.1 Introduction

Networks are becoming ubiquitous as they can represent many real-world relational data, for instance, information networks, molecular structures, telecommunication networks, and protein-protein interaction networks.

A *Network* is a collection of entities and feasible connections between them. A Network is most commonly represented using a *Graph*. Network analysis is carried out by analysing the underlying graph. Even in non-network applications where the data are not explicitly linked, it is possible and helpful to represent the data in the form of a network/graph. For example, in probabilistic graphical models, the data is visualized as a graph.

A *graph* is a non-euclidean data structure which is represented by $G = (V, E)$, where V is the set of nodes and E is the set of edges. The *nodes* represent the network entities and *edges* represent the connections between the entities. An edge $e_{ij} \in E$ between two nodes v_i and v_j is represented by a pair of the nodes (v_i, v_j). For example, in a social network, each node v_i represents a user, and an edge (v_i, v_j) represents whether user v_i is friends with user v_j or not. In protein-protein interactions network, nodes represent proteins and edges represent interactions between these proteins. Also, some real-world graphs have an associated set of attributes, where each node $v_i \in V$ is associated with an attribute vector $x_i \in \mathbb{R}^D$, D is the dimension of each attribute vector. Further, graphs can also have an associated node labels set L, where $y_i \in L$ is the label of node v_i, which specifies the class of the node.

Analysis of these networks provides advantages in many fields such as *recommendation* (recommending friends in a social network), biological field (deducing connections between proteins for treating new diseases), *community detection* (grouping users of a social network according to their interests), etc. by leveraging the latent information of networks. Hence, network analysis is gaining prominence. But the high dimensional, irregular graph data imposes challenges for machine learning tasks which led to the development of many representation learning techniques.

M. Aggarwal and M. N. Murty, *Machine Learning in Social Networks*,
SpringerBriefs in Computational Intelligence,
https://doi.org/10.1007/978-981-33-4022-0_2

2.2 Networks Represented as Graphs

Any network can be easily represented as a *Graph*, defined in Sect. 2.1, which facilitates modeling data items and relations among them.

Many variants of *Graphs* are possible including:

- *Heterogeneous Graphs*: The nodes or/and edges of such graphs can be of various types and each type must be handled differently.

- *Homogeneous Graphs*: Contrary to heterogeneous graphs, nodes and edges are instances of a single type.

- *Directed and Undirected Graphs*: Directed graphs have ordered pairs of vertices, and each edge has a starting point (head) and an ending point (tail), and information flows from head to tail. In contrast, an edge in an undirected graph can be traversed in both directions representing a symmetric relation. Further, an undirected edge can be replaced with two directed edges. In a directed graph, an edge between nodes v_i and v_j is represented by (v_i, v_j), whereas in the undirected graph, it can be written either way i.e., (v_i, v_j) or (v_j, v_i).

- *Dynamic Graphs*: Some real-world networks might evolve over time. For example, in social networks, new users can be added, or new interactions might occur between existing users. This leads to the addition or removal of nodes or edges, respectively, and hence these graphs are called *dynamic*. On the contrary, graphs which do not change over time are known as *static graphs*.

- *Knowledge Graphs*: A Knowledge Graph is a directed, multi-relational graph where an edge is represented in (head entity (h), relation (r), tail entity (t)) form, which means that *h* is related to *t* through *r*. For instance, (Star Trek, Genre, Science Fiction).

- *Hypergraphs*: They are the generalization of undirected graphs in which edges are over subsets of two or more vertices. Formally, a hyper-graph H is a pair $H = (X, E)$ where X is a set of elements called nodes or vertices, and E is a set of non-empty subsets of X called hyper-edges.

All these variants might contain useful auxiliary information such as vertex attributes and/or vertex labels, in addition to the connectivity/structural information.

A toy example of an undirected and homogeneous graph with 7 nodes and 9 edges is depicted in Fig. 2.1. To understand some important properties of a graph, let us observe the same.

1. This example illustrates an undirected and static homogeneous graph. Therefore, edges can be represented in both directions. For example, an edge between *a* and

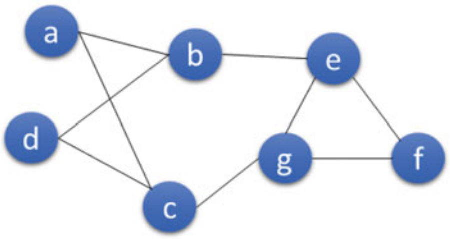

Fig. 2.1 A Toy network: Circles ($a, b, \ldots, g$) are the nodes of the graph while black lines denote the edges

b can be denoted as (a, b) or (b, a). In a friendship network, if u_1 and u_2 *are connected* then stating that u_1 *is a friend of* u_2 *is the same as* u_2 *is a friend of* u_1.

2. *Neighbours* are those nodes that are connected by an edge. For example, node a and node b are connected by an edge and hence are called neighbors (also called adjacent nodes) of each other.
3. *Common Neighbor:* Further observe that both nodes a and b have an edge to node c, that is why c is their *common neighbour*.
4. *Degree* of a node is the total number of incident edges on that node in an *undirected graph*. The degree of node a is 2. Further, the sum of the degrees is equal to 2 times the number of edges:

$$\sum_{i=1}^{|V|} deg(v_i) = 2|E| \tag{2.1}$$

where $|V|$ is the total number of nodes in the graph.

5. In directed graphs, the degree of any node v is the sum of indegree and outdegree of v, where *indegree* is the number of incoming edges on v and *outdegree* is the number of edges leaving v. Also, the total indegree of the graph is equal to the total outdegree of the graph.

A graph can be represented in numerous ways. All variants, as described in Sect. 2.2, need different representation schemes. In this book, our focus is on undirected/directed, static homogeneous graphs. Next, we discuss the two most commonly used methods to represent graphs on the machine.

2.3 Data Structures to Represent Graphs

2.3.1 Matrix Representation

Adjacency Matrix (A): It is a square matrix where number of rows and number of columns are the same as the number of nodes in the graph, i.e. the dimension of A is $N \times N$ where N is the number of nodes. Each (i, j)th entry in the matrix indicates

Table 2.1 Adjacency matrix for the graph in Fig. 2.1

From/To	*a*	*b*	*c*	*d*	*e*	*f*	*g*
a	0	1	1	0	0	0	0
b	1	0	0	1	1	0	0
c	1	0	0	1	0	0	1
d	0	1	1	0	0	0	0
e	0	1	0	0	0	1	1
f	0	0	0	0	1	0	1
g	0	0	1	0	1	1	0

presence or absence of an edge between nodes v_i and v_j. If $A_{i,j}$ is 1 then nodes v_i and v_j are connected by an edge otherwise 0. If the graph is weighted then each entry of the matrix will store the weight of the corresponding edge.

Table 2.1 describes the matrix A for graph in Fig. 2.1. Some important points to observe from A are:

- Each node v_i of a graph is given an index i and ith row of matrix A corresponds to v_i.
- The graph in Fig. 2.1 has 7 nodes; therefore, dimension of matrix A is 7×7.
- As the graph is undirected, A is a symmetric matrix. For a directed graph, the adjacency matrix may not be symmetric.
- *Degree* of a node for an undirected graph can be computed by taking the sum of entries in either the respective row or the column of matrix A. Degree of node a is 2 as

$$\sum_{i=0}^{6} A_{i,0} = \sum_{i=0}^{6} A_{0,i} = 2 \tag{2.2}$$

- To store matrix A, $O(N^2)$ space is required, where N is the number of nodes.
- Observe that all diagonal entries are 0. If a graph has a self loop on a vertex v_i, then A_{ii} will be equal to 1, and each self loop contributes 2 to its degree (incoming and outgoing is the same vertex).
- Also note that matrix A of a network abstracts paths of length 1, $A \times A$ gives paths of length 2 and so on.

2.3.2 Adjacency List

Another way to represent a graph is by using its Adjacency List (adjlist). It is an array of linked lists where the ith list, adjlist[i], stores references to every neighbour of node v_i (Fig. 2.2).

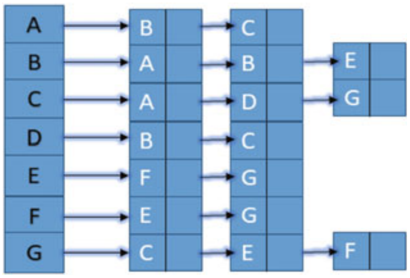

Fig. 2.2 Adjacency list of the network in Fig. 2.1

- Each node v_i of a graph is given an index i and the respective linked list stores its adjacent nodes. As can be observed from Fig. 2.1, node a (given 0th index) has nodes b and c as its neighbors and hence adjlist[0] has pointer to the linked list (an array can also be used instead of linked list) which stores references to nodes b and c. Also, the number of linked lists is equal to the number of nodes.
- Another interesting point is that if $(v_i, v_j) \in E$ in an undirected graph, then the list of v_i will have an entry for v_j and vice versa, whereas if it is an edge in a directed graph, then only v_i will have an entry for v_j in its list.
- For weighted graphs, each node entry will have two items, one for node index and the other one for the respective edge weight.
- A further observation is that the length of a list tells the degree of the respective node in undirected graphs.

2.4 Network Embeddings

Because of the prominence of networks in many real-world problems, network analysis is gaining importance in many disciplines. Networks contain a pool of information needed by all end stream tasks.

Due to this, analysis schemes depend substantially on network representations (also known as *encodings or embeddings*). Furthermore, structural information, which may be intractable, has to be inscribed in the low dimensional vector space. For example, classification of a node requires the global position and the local neighborhood structure of the node; link prediction task needs to capture the similarity between two or more nodes. But in some application domains, the network structure may not be apparent.

Therefore, we need an efficient and optimal way to convert the raw non-euclidean high dimensional network data to a vector such that implicit or explicit relations/properties present in the network are protected in the vector space. For example, distance between nodes in a network should be preserved in the embedding space. *Embedding learning* is also called *representation learning.*

Network Representation Learning: Given a network represented by a graph *G=(V,E)* and some side information related to G such as a set L of discrete labels to label the nodes and a matrix of vertex attributes X. The aim is to learn a mapping, $g_v : V \mapsto Z$, that maps each node v_i of G to a vector z_i of dimension d known as embedding.

- Conventional ways of using adjacency vector for representing nodes are not appropriate as large scale data makes computation intractable in high dimensional spaces. Also, they capture only first-order dependencies and are incapable of including higher-order relations of a network.
- However, network representation learning methods learn explanatory embedding vectors because of which many machine learning analysis tasks such as recommendation, link prediction, node and graph classification, community detection, and visualization can be efficiently tackled using these embeddings.
- Important properties that embeddings must encode are explained below.
  - As discussed earlier, some real-world graphs are augmented with node attributes that help network representation learning techniques to learn more discriminative embeddings as these approaches also capture the attribute level similarity between nodes.
  - Other common properties of interest are the relationships of various order of proximities between nodes.
    - The first-order proximity captures edges of a graph. Thus, if v_i, v_j are connected by an edge then the first-order proximity is 1 between v_i and v_j. And it is 0 for non-adjacent pairs of nodes. This information can easily be gathered from the adjacency matrix of the graph.
    - Two-hop relations are described by the *second-order proximity* which is measured by the number of 2-hop paths between v_i and v_j. These paths between a pair of nodes can be calculated using the second-order adjacency matrix A^2.
    - Similarly, *higher-order proximity* captures node pairs with q-hop paths where q is greater than or equal to 3. This is determined by q-step transition probability i.e., the probability of landing on node v_j at qth step of a random walk starting from node v_i. As the order of proximity increases, the ability to capture global structure also increases.
  - Moreover, many nodes of a graph share the common responsibility such as nodes playing the role of a hub node. This property is also known as the *structural equivalence* property.
  - Another important principle is homophily, i.e. the nodes of a graph form a community structure. All vertices in a single community share some common interest or property. For example, in citation networks, all papers in a single community are on the same or a similar topic.
  - An acceptable representation should exhibit a proper balance of both of these equivalences i.e., embeddings of nodes from the same community should be

similar to each other and the nodes with similar structural roles should also be embedded closer to each other.

2.5 Experimental Datasets and Metrics

Many Benchmark datasets are being used to evaluate the quality of the network embeddings or performance of algorithms on various downstream tasks.

- Some datasets contain information of a single graph, e.g. adjacency matrix, node attributes, node labels, etc. They are used for node or edge level analysis, such as node classification, and link prediction.
- Some datasets are a collection of graphs and are used to evaluate the efficiency of the graph level embeddings on graph level tasks, such as graph classification, graph clustering, etc.

This section briefs some of the most commonly used datasets and the evaluation metrics for various downstream tasks.

2.5.1 Evaluation Datasets

Many datasets are publicly available for measuring the effectiveness of algorithms and comparing them with the state-of-the-art approaches depending on the performance on various downstream tasks. This section discusses some of the most frequently used datasets.

1. *Citation networks*:
   These datasets are collections of scientific, academic publications with nodes denoting authors or papers and edges representing author-coauthor or paper-paper citation relations. Examples of citation networks include Cora, Citeseer, PubMed datasets. Below is a brief description of these datasets (Table 2.2).
   - Cora, Citeseer, PubMed are unweighted and directed networks and have node attributes that denote the contents of the papers or authors.
   - The number of distinct class labels varies across the datasets. Cora has seven class labels, Citeseer has six distinct node labels, while PubMed has three distinct node labels.
2. *Biological networks*:
   - *Single graph Dataset:* A PPI (Proteins-Proteins Interaction) dataset is from this category whose nodes represent proteins and edges represent the existing physical interactions. It has 40 different node labels, with each class denoting some biological state.

- *Collection of networks (graphs):* Each dataset has information for multiple networks (graphs). Some frequently used datasets are MUTAG, PROTEINS, NCI1/NCI109, ENZYMES, etc. All these datasets have an adjacency matrix for each graph, graph identifiers for all the nodes, and graph labels for all the graphs. Further, this information can be supplemented by node attributes, edge labels, and graph attributes (Table 2.3).
  - *MUTAG:* It is a collection of chemical compounds having different mutagenic behavior on a bacterium. It contains information of 188 graphs categorized into two classes.
  - *PROTEINS:* This dataset is a collection of proteins, where graphs are secondary structure elements. There is an edge between two nodes if the nodes are in any sequence of amino acids. The number of graphs is 1113, and the number of graph labels is two.
  - *ENZYMES:* Each graph in the set is a protein tertiary structure. It is a collection of 600 graphs, with each graph categorized into one of the two classes.

Other bioinformatic datasets are PTC, FRANKENSTEIN, etc. The details of these datasets can be found at (https://bit.ly/39T079X). Table 2.3 contains a high-level summary of these datasets.

3. *Social Networks:*
   - *Single graph Dataset:* Some commonly used datasets are YouTube, Flickr, BlogCatalog, where nodes correspond to the users of that social website, and edges describe the relations between users of the website. For instance, nodes of the BlogCatalog network represent bloggers. Important statistics are provided in Table 2.4.
   - *Collection of networks (graphs):* Datasets such as IMDB-BINARY, IMDB-MULTI, COLLAB, REDDIT (Binary and Multi), etc. are the most commonly used graph level social network datasets. The details of these datasets can be found at (https://bit.ly/39T079X). Refer to Table 2.5 for a high-level description of these datasets.
4. *Collaboration Networks*: Arxiv is a collaboration network formed from the ArXiv website, and edges represent the co-author relations. Papers are only from a single field, and therefore the corresponding node labels are absent. Missing node label information makes this dataset suitable for link prediction.

There are many other types of datasets, e.g., Language Networks (Wikipedia), Communication Networks (Enron Email Network), etc.

2.5.2 Evaluation Metrics

As discussed, the embeddings are used for various downstream tasks (detailed discussion in Sect. 2.6). The performance on these tasks throws light on the algorithm's

Table 2.2 Citation networks for node level experiments

Dataset	#Nodes	#Labels	Attributes
Cora	2,708	7	Yes
Citeseer	19,717	6	Yes
PubMed	3,312	3	Yes

Table 2.3 Different biological datasets used in graph level experiments

Dataset	#Graphs	#Max Nodes	Avg.#Nodes	Avg. #Edges	#Labels	Attributes
MUTAG	188	28	17.93	19.79	2	NO
PTC	344	64	14.29	14.69	2	NO
ENZYMES	600	125	32.63	62.14	6	Yes
PROTEINS	1113	620	39.06	72.82	2	Yes
DD	1178	5748	284.32	715.66	2	NO
NCI1	4110	111	29.87	32.30	2	NO
NCI109	4127	111	29.68	32.13	2	NO
FRANKENSTEIN	4337	214	16.90	17.88	2	YES

Table 2.4 Social networks for node level experiments

Dataset	#Nodes	#Labels	Attributes
BlogCataloga	10,312	39	NO
Flickrb	80,513	195	NO
YouTubeb	1,138,499	47	NO

Table 2.5 Social graph level datasets

Dataset	#Graphs	#Max Nodes	Avg. #Nodes	Avg. #Edges	#Labels	Attributes
IMDB-BINARY	1000	136	19.77	96.53	2	NO
IMDB-MULTI	1500	89	13.00	65.94	3	NO
COLLAB	5000	492	74.49	2457.78	3	NO
REDDIT(BINARY)	2000	3782	429.63	497.75	2	NO
REDDIT(MULTI)-12K	11929	3782	391.41	456.89	11	NO

efficacy and helps in comparing different algorithms. This section details the metrics which are popularly used to evaluate the performance of the embeddings through algorithms on different downstream tasks.

- *Classification Accuracy* is the simplest metric which tells how many correct predictions a model makes i.e., $\frac{\#Correct\ Predictions}{\#Total\ Samples}$. This measure may not be useful when the data has *class imbalance*, that is most of the training patterns are from one class and the other class(es) have a very small number of training patterns.

- *F1 score* is the weighted average of precision (P) and recall (R) and its value lies between 0 and 1, with 1 being the highest score i.e., a model needs to maximize the F1-score.

$$F1_{micro} = 2 * (P * R)/(R + R) \tag{2.3}$$

This is also known as *F1-micro score*. Another score is *F1-macro*, which is defined as

$$F1_{macro} = \frac{\sum_{y \in L} F1(y)}{|L|} \tag{2.4}$$

Here $y \in L$ is the node label, $F1(y)$ is the $F1$-score for label y and $|L|$ is the number of distinct node labels.

  – *Precision and Recall:*

    · *Precision* calculates how precise the model is or how many are actually positive (true positive) among all the predicted positives. This metric is useful when misclassifying negative sample costs more. Precision is defined as:

$$Precision(P) = \frac{\#True\ Positives}{\#True\ Positives + \#False\ Positives} \tag{2.5}$$

    · *Recall* calculates how many samples from the positive class the model can predict correctly. This is used when misclassifying a positive class sample costs more to the user. Recall is described as:

$$Recall(R) = \frac{\#True\ Positives}{\#True\ Positives + \#False\ Negatives} \tag{2.6}$$

  Here, true positives are those inputs that belong to the positive class and are classified correctly. False positives are those samples which belong to the negative class but are misclassified as positives. Similarly, true negatives belong to the negative class and are correctly classified, while false negatives belong to the positive class and are misclassified.

- *Precision at k (Pr@k)* : Instead of evaluating the model with respect to all the samples, it calculates the correct number of predictions in only the top k predicted edges:

$$Pr@k(i) = \frac{|\{v_j | v_i, v_j \in V, (v_i, v_j) \in E, rank(v_j) \leq k\}|}{k} \tag{2.7}$$

Here V is the set of nodes, E is the set of edges, $v_i, v_j \in V$ are the nodes, and rank(.) is the rank of the node. This metric is used to measure the efficiency of algorithms on the link prediction task.

- *NMI (Normalized Mutual Information)*: It is a normalized (i.e., between 0 and 1) mutual information (MI) score with 0 denoting no MI and 1 being the perfect correlation. This score is permutation invariant; thus, the score will remain the same for any permutation of the cluster labels. NMI is defined as:

$$NMI(L, W) = \frac{I(L; W)}{\frac{[H(L)+H(W)]}{2}} \tag{2.8}$$

Here L is the set of actual labels (ground truth clusters based on the node labels), W is the set of predicted labels based on the predicted clusters, $H(.)$ is the entropy function, and $I(., .)$ is mutual information.

– *Entropy of class labels (H(Y))* talks about the uncertainty and is calculated using the following equation:

$$H(L) = -\sum_{i \in L} P(L = i) \times \log P(L = i) \tag{2.9}$$

– Entropy of cluster labels (H(W)) is also calculated similarly:

$$H(W) = -\sum_{i \in W} P(W = i) \times \log P(W = i) \tag{2.10}$$

Here P(W=i) is given by $\frac{\#Samples\ in\ cluster\ i}{Total\ Number\ of\ Samples}$.

– The next equation describes how to calculate the MI between the class labels and the cluster labels, which denotes the entropy reduction of class labels when cluster labels are given.

$$I(L; W) = H(L) - H(L|W) \tag{2.11}$$

Here H(L|W) is the conditional entropy of class labels, and for each cluster *i*, it is calculated using the following equation:

$$H(L|W = i) = -P(W = i) \sum_{l \in L} P(L = l|W = i) \log(P(L = l|W = i)) \tag{2.12}$$

Here P(L=l|W=i) is the probability of getting sample with label l in cluster i.

Putting all these values together, we can get the final NMI score using Eq. 2.8.

- *Purity* is a simple clustering measure that determines how many samples are correctly placed after clustering. All samples in a cluster are given a single label according to the most frequent label in that cluster. The fraction of correctly labeled samples to the total samples is known as the Purity.

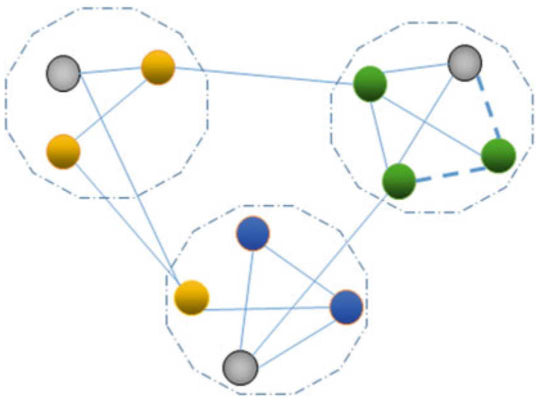
Fig. 2.3 A Toy Network that depicts the results of Machine Learning tasks. (1) All nodes are colored according to the node labels such as all blue color nodes have same node label. All grey colored nodes are unlabeled. (2) Dotted circles denote a cluster. All nodes within a dotted circle belong to the same cluster or community. (3) Dotted lines are the predicted links

2.6 Machine Learning Downstream Tasks

One of the important factors behind learning low-dimensional network embeddings is that the same embedding can be exploited to deal with a host of *machine learning* (ML) tasks. These downstream ML tasks are performed on top of the trained embeddings or in an end-to-end fashion while learning embeddings.

- These ML tasks include *classification, clustering, link prediction, visualisation, and reconstruction*.
- Any off-the-shelf machine learning algorithm can perform the tasks mentioned above using the learned embeddings as input features.
- The performance on these machine learning tasks is further used for a fair comparison of approaches and as a measure to evaluate the quality of the learned embeddings.

In this section, we will explain these tasks with the help of Fig. 2.3 and provide their respective evaluation measures.

2.6.1 Classification

This is the most commonly used downstream task to justify the effectiveness of the trained embeddings.

- In practice, some networks are augmented with node labels or edge labels or graph labels categorizing these network entities into distinct categories. For example, in a citation network, node labels correspond to the field of research of the corresponding paper or author.
- Consider the example network shown in Fig. 2.3.
  - It has 11 nodes out of which 8 of them are already labeled.

  - There are 3 nodes that are colored green, three others are colored yellow and two nodes are colored blue. Nodes with the same color are from the same class.
  - There are 3 nodes that are colored grey and they are unlabeled.

- In real-world data also, some of the nodes/edges/graphs are unlabelled. Hence, the task of classification aims to classify these unlabeled entities into their respective classes by making use of the existing structure and information that encompasses the network.
- If the learning process is (semi) supervised, which uses labels during training, then learned embeddings are more discriminative as they are trained in an end-to-end fashion with the end task being the classification task.
- However, in unsupervised learning, first, the embeddings are learned, and then popular classifiers, e.g. logistic regression is used with the learned embeddings forming the input features.
- For example, in Fig. 2.3, all grey colored nodes are unlabeled, and the classification process will classify these nodes into appropriate classes. We expect the grey colored node in the neighborhood of the three green colored nodes to be classified so that it is also colored green.
- Evaluation metrics for analyzing the classification performance typically are *classification accuracy* and *F1 score*.

2.6.2 Clustering

Many nodes share the same property in a network, and together they form a soft cluster or community. For instance, in a social network, users with interest in the same domain will form a community/cluster; in a citation network, papers having similar research domains form a single group. The process of finding such soft clusters is called *community detection* in the context of network analysis.

- This task aims to partition nodes or (sub)graphs into various groups in a way that similarity between samples within a cluster is maximum, whereas samples from different clusters are dissimilar.
- The partition is typically soft; it can be hard if the application demands.
- This task is very important in the biological field (proteins-proteins network, drug-disease network) to find related and interacting drugs or associated diseases.
- In the example network shown in Fig. 2.3, there are three clusters based on connectivity among the nodes; connectivity provides the similarity here. These are shown using dotted circles.
- Similarly, in a set of graphs datasets, many graphs share the same property(ies) and belong to the same cluster. For example, graphs that have a similar clique structure can be grouped.
- Any generic clustering algorithm such as K-means, K-means++, or LDA can be used after learning node or (sub)graph embeddings to obtain communities.

- It is quite similar to the community detection task, and if the learned representations are discriminative, they should be able to detect appropriate community structure exhibited by the graph.
- The embedding's performance on the clustering task is measured by purity, ARI, and normalized mutual index metrics by leveraging ground truth clusters based on the node labels.

2.6.3 Link Prediction (LP)

It aims to predict new connections that are likely to occur in the future based on the existing relations in the graph and the network properties.

- This will further throw light on how the network evolves.
- Link prediction can also infer the missing connections (edges) in the network.
- Major applications of link prediction are in recommender systems, social networks, biological networks. In social networks, one can use LP to predict friendship among users of the network and recommend the friendship connections. Biological networks (drug-disease networks) take advantage of this task to expand the biological dataset and infer new diseases or treatments via predicting interactions between entities.
- For example, all the dotted lines between a pair of nodes in Fig. 2.3 are the predicted links.
- For evaluation, some links (connections) from the given graph are deleted followed by learning node embeddings using the new graph. These embeddings are then used to predict these deleted edges.
- The proficiency of embeddings on how well they can support the network evolution depends on the quality measured by the metrics AUC and precision.

2.6.4 Visualization

This task helps in data mining and analyzing real-world high dimensional data visually by projecting it into two or three dimensions.

- Once low-dimensional embeddings are trained, which encode the network structure, any of the available visualization tools can be used.
- For instance, t-SNE takes as input the learned encodings and projects them into 2D or 3D space. Such an easy to visualize plot will help in inferring clusters or communities.
- Principal Components can also be used for mapping the embeddings into a lower dimension space for easy visualization.
- These visualizations provide insights into the quality of the embeddings based on whether the nodes from the same class (or belonging to the same community) are

close to each other. Also, nodes with different class labels need to be far away in the projected space.

2.6.5 Network Reconstruction

- It aims at reconstructing the actual network (graph) using similarity between each pair of node embeddings.
- The similarity score between embedding vectors determines the similarity between two nodes and is used to infer the edge between them.
- If the learned embeddings are discriminative enough, then the underlying similarity function should detect the original links present in the network, which determines the quality of these learned representations.
- In the process, it will be interesting to look at existing edges that need to be deleted based on the low similarity between the embeddings of the end vertices.

2.7 Embeddings Based on Matrix Factorization

We have seen that a popular way of representing a network graph $G = (V, E)$ is to use its adjacency matrix A.

- The $N \times N$ matrix A captures the structure/connectivity information. Each row of the matrix is a vector of size N based on the presence/absence of links of a node to all the nodes in the network. In a practical setting N could be very large.
- In addition to the connectivity, if each node has some content/attribute information, then a matrix X of size $N \times D$ is used. Here each node is viewed as a vector of size D, where D is the size of the vocabulary behind the content, which could also be large.
- So, we need to represent both the structure and content data in a low-dimensional space to facilitate efficient and accurate machine learning on the network data.
- *Matrix factorization* is one of the well-known techniques to reduce the dimensionality. This may be viewed as follows:
  - We can factorize the adjacency matrix A as $A = BC$ where A is a $N \times N$ adjacency matrix, B is a $N \times K$ matrix, and C is a $K \times N$ matrix.
  - Similarly the matrix X can be factorized as $X = GH$ where X is of size $N \times D$, G is a matrix of size $N \times K$, and H is a matrix of size $K \times D$.
  - Typically the value of K is much smaller than both N and D to facilitate dimensionality reduction.
  - If the structure and content in the network are in perfect agreement with each other, then their low-dimensional representations will also be agreeing with each other. This could be to the extent that the two matrices B and U are equal which can simplify the factorizations. In practice, B and U could be different because

of noise in the correspondence. In such a case one can aim to minimize some difference between the two matrices.

There are several well-known techniques for factorization of matrices. These are considered in the next few subsections.

2.7.1 Singular Value Decomposition (SVD)

Singular value decomposition (SVD) is the most popular matrix factorization technique. Either a square matrix ($N \times N$ matrix) like A or a rectangular matrix ($N \times D$ matrix) like X can be factorized using SVD.

- In general any real matrix S of size $p \times q$ can be decomposed into

$$S_{p\times q} = U_{p\times K}\Sigma_{K\times K}V^T_{K\times q}, \ \mathit{where}$$

  - V^T is the transpose of the matrix $V_{q\times K}$.
  - The non-zero eigenvalues of SS^T and S^TS are the same; it is possible that some of these eigenvalues are repeated. If $p > q$ then SS^T will have at least $(p - q)$ 0 (zero) eigenvalues. Similarly, if $q > p$ then S^TS will have at least $(q - p)$ 0 (zero) eigenvalues. The common eigenvalues are non-negative.
  - Σ is a diagonal matrix of size $K \times K$. Its diagonal entries are the positive square roots of the K largest eigenvalues of either SS^T or S^TS, where $K < min(p, q)$.
  - These diagonal entries of Σ are called the *singular values* of S. They are non-negative real numbers, if S is real, and are typically arranged in non-increasing order.
  - Columns of U are the K eigenvectors of SS^T and columns of V are the K eigenvectors of S^TS. These K vectors are the orthonormal eigenvectors associated with the top K common eigenvalues.

- So, SVD can be used for matrix factorization. Further, it can be used to obtain the *principal components* (PCs) of the data present in the form of the row vectors of either A or X matrices associated with the graph representing any network.
- If the rows of the matrix S are normalized so that their mean is zero, then the eigenvectors of S^TS (columns of V) are the principal components.
- If the matrix to be factorized is symmetric, then we can decompose it using orthogonal matrices. For example, the adjacency matrix A will be a symmetric matrix if the graph is undirected. In such a case we have

$$A_{N\times N} = P_{N\times K}D_{K\times K}P^T_{K\times N},$$

  where P is an orthogonal matrix, that is $P^T = P^{-1}$.

2.7.2 Matrix Factorization Based Clustering

It is possible that some of the entries in U or V or both can be negative. A consequence of this is that even the PCs can have negative entries.

- We can illustrate this using a simple example. Consider the dataset shown in Table 2.6.
  - There are four objects represented as two-dimensional patterns in the table; each is described by the Volume of the object and its Price both in respective units. For example, the first object has 1 unit of volume and 8 units of price.
  - The sample mean of the 4 points is (2,5). So, the zero mean normalized data is given by the matrix

$$Z = \begin{bmatrix} -1 & 3 \\ 1 & -3 \\ -1 & 3 \\ 1 & -3 \end{bmatrix}$$

  - The matrix $Z^T Z$ is given by

$$Z^T Z = \begin{bmatrix} 4 & -12 \\ -12 & 36 \end{bmatrix}$$

  - The eigenvalues of the matrix $Z^T Z$ are 40 and 0 and its orthonormal eigenvectors are $\begin{pmatrix} \frac{1}{\sqrt{10}} \\ \frac{-3}{\sqrt{10}} \end{pmatrix}$ and $\begin{pmatrix} \frac{3}{\sqrt{10}} \\ \frac{1}{\sqrt{10}} \end{pmatrix}$.
  - These two vectors are the two PCs in that order and they are orthogonal to each other and there is a negative entry in the first PC.
  - This example clearly illustrates that PCs can have negative entries.

- However, there could be applications where we require only non-negative entries in the factor matrices. Clustering or community detection is one such example. We can explain using the example data in Table 2.6.
  - Note that in the two-dimensional space characterized by Volume and Price, pattern 1 and pattern 3 are identical. Similarly, pattern 2 and pattern 4 are also identical.
  - So, if we want to assign these 4 patterns into two clusters, then pattern 1 and pattern 3 are in one cluster and the remaining two patterns are in the other cluster.
  - Such a cluster structure may be realized using the matrix factorization, $X_1 = G_1 H_1$, that is exemplified by

$$\begin{bmatrix} 1 & 8 \\ 3 & 2 \\ 1 & 8 \\ 3 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 8 \\ 3 & 2 \end{bmatrix}$$

- Observe that G_1 matrix (the 4×2 matrix in the RHS) is the *cluster assignment matrix*. The two columns of G_1 correspond to the two clusters. Pattern 1 and pattern 3 are assigned to cluster 1 and the corresponding entries in column 1 are 1. Similarly, pattern 2 and pattern 4 are assigned to cluster 2 and the second column in matrix G_1 indicates this assignment.
- It is important to note that the entries in G_1 and H_1 are nonnegative. This property is essential here.
- Note further that the matrix H_1 is the *cluster description matrix*. The two rows of H_1 describe the two clusters. Here, each cluster is described by its centroid.
- In this simple example, both the patterns in each cluster are identical. In general, a centroid-based clustering algorithm like the K-means clustering algorithm describes each cluster by its centroid. In such a case, the ith row of matrix H_1 will be the centroid of the ith cluster, for $i = 1, \ldots, K$.
- For example, if pattern 3 and pattern 4 in Table 2.6 are changed to (1,6) and (5,2) respectively, then the factorization, for $X_2 \approx G_2 H_2$ is given by

$$\begin{bmatrix} 1 & 8 \\ 3 & 2 \\ 1 & 6 \\ 5 & 2 \end{bmatrix} \approx \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 7 \\ 4 & 2 \end{bmatrix}$$

  where X_2 is the modified version of X_1 and G_2 and H_2 are respective cluster assignment and cluster description matrices.
- Note that pattern 1 and pattern 3 are assigned to cluster 1 and the remaining two patterns are assigned to cluster 2. In this case, $G_2 = G_1$ and H_2 has the cluster centroids (1,7) and (4,2) as its rows.
- Observe further that the factorization of X_1 is exact whereas the factorization of X_2 is approximate.
- In general, any hard clustering output can be abstracted by such a non-negative matrix factorization where $X \approx GH$ where each row of G will have one entry 1 and the remaining $K - 1$ entries 0.
- If the entry G_{ij} in the ith row and the jth column of G is 1, then it indicates that the ith pattern, that is the ith row of X is assigned to cluster j.
- All other entries in the ith row of G will be 0; that is $G_{ik} = 0,\ \forall k \neq j$. This is because in hard clustering a pattern is assigned to one and only one cluster.
- Further, $H(i)$, the ith row of H represents the ith cluster. It could be the centroid in the case of K-means algorithm; but in general it could any vector that is a suitable *representative of the* ith *cluster*.

Every *hard clustering* output can be viewed as leading to such a matrix factorization. For example, the output of spectral clustering also can be represented in terms of matrix factorization.

Table 2.6 Example data matrix

Pattern	Volume	Price
1	1	8
2	3	2
3	1	8
4	3	2

2.7.3 *Soft Clustering as Matrix Factorization*

There are several applications where a natural requirement is to assign a pattern to more than one cluster. For example, a document may share more than one topic/cluster; for example, it may be dealing with both *sports* and *politics*. In such a case, we require the clustering to be soft.

- *Topic models* like the latent Dirichlet allocation (LDA) are probabilistic and they assign a document to more than one topic/cluster.
- In such a case, in the approximation of X as GH, the ith row of G has one or more non-zero entries. The entries in any row of G add upto 1. G_{ij} could be viewed as the probability that the ith document belongs to the jth topic/cluster.
- The ith row of H is the probabilistic description of the ith topic/cluster. In fact a topic is an assignment of a probability value to each term in the vocabulary present in the collection of documents.
- It is not just the LDA alone. Every soft clustering output could be represented using an appropriate factorization of matrices.
- For example, probabilistic latent semantic indexing ($PLSI$) is one such example. The output of $PLSI$ can be abstracted as

$$S_{p\times q} = U_{p\times K} Z_{K\times K} V_{K\times q}$$

  - Here the $p \times q$ matrix S represents p documents using q vocabulary terms, where the input is a collection of p documents and the vocabulary size is q.
  - The matrix U is of size $p \times K$. It describes p documents using K topics/soft clusters.
  - The matrix V of size $K \times q$ is a description of the K topics using the q vocabulary terms. The ith row of V describes the ith topic for $i = 1, \ldots, K$ and each row is a q-dimensional vector with one or more probability entries.
  - The $K \times K$ matrix Z is a diagonal matrix that describes the strength of each of the topics. The diagonal entry in the ith row and the ith column of Z, that is Z_{ii} indicates the importance of topic i in the collection.

There could be applications where the factorization could be deterministic rather than probabilistic.

2.7.4 Non-Negative Matrix Factorization (NMF)

One of the well-known matrix factorization approaches that is deterministic is the NMF. Here, a matrix X is approximated using two factors G and H.

- It is viewed as minimizing the squared euclidean norm between X and GH. So, It is based on
$$||X - GH||^2.$$
- This norm is the sum of element-wise deviations or the squared deviations between X_{ij} and $(GH)_{ij}$ are added over all the elements, that is for $i = 1, \ldots, N$ and $j = 1, \ldots D$.
- We find G and H so that $||X - GH||^2$ is minimized under the constraints that $G_{ij} \geq 0$ and $H_{ij} \geq 0$ for all i and j.
- If X and G are known then finding H is a well-behaved convex optimization problem. Similarly, if X and H are given, then finding G is also a convex optimization problem.
- So, the problem of finding G and H given X is solved by using an *alternating minimization* process. Using X and some initial G, H is computed. Using X and the obtained H, G is updated. This process goes on till some termination condition is satisfied.
- The solution obtained by this optimization is locally optimal. Note that several of the factorization schemes can give us only a local optimum. These include the K-means clustering, LDA, $PLSI$, and NMF.
- On the other hand SVD and the orthogonal decomposition are deterministic and give us the exact factorization.
- The motivation behind NMF was that the factors G and H provide information about the presence/absence of parts of objects in the data.
- Consider the example matrix factorization given by

$$\begin{bmatrix} 1\,1\,0\,0\,1\,1 \\ 1\,1\,0\,0\,1\,1 \\ 0\,0\,1\,1\,0\,0 \\ 0\,0\,1\,1\,0\,0 \end{bmatrix} = \begin{bmatrix} 1\,0 \\ 1\,0 \\ 0\,1 \\ 0\,1 \end{bmatrix} \begin{bmatrix} 1\,1\,0\,0\,1\,1 \\ 0\,0\,1\,1\,0\,0 \end{bmatrix}$$

- This illustrates NMF. Note that the 4×6 matrix in the LHS is a rank 2 matrix. It clearly shows two linearly independent row vectors, (110011) and (001100) and the two linearly independent column vectors $\begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$.
- The factorization aptly captures this by exploiting the fact that the rank of the matrix is 2 ($K = 2$).
- In general, the value of K is upper bounded by the minimum of N and D where X is of size $N \times D$.

- Further note that the independent columns form the G matrix and the independent rows form the H matrix. These basis vectors may be viewed as parts of the objects or patterns present in X.
- Further, note that there are two hard clusters and the assignment of the first two vectors to the first cluster and the remaining two rows to the second cluster are depicted by the 4×2 matrix in the example.
- Similarly, the two rows of the 2×6 matrix, in the RHS of the example, capture the description of the two clusters.
- A fundamental observation from linear algebra is that *row rank* = *column rank* = *rank* of any matrix.
- This implies that clustering and dimensionality reduction have an excellent correspondence as depicted in the example factorization.

Even though matrix factorization is very popular in several machine learning applications, its main drawback is that it is computationally not very attractive to deal with large-scale network datasets. An alternative is based on *random walks*. We consider in the next section, *word2vec* that offered the basic platform for several state-of-the-art network embedding schemes that use random walks.

2.8 Word2Vec

It is the most influential tool in terms of its impact on network embedding schemes. It deals with representing each word as a vector. Neural network models require real-valued vectors as input. So, a string or a word needs to be converted into a vector of numbers to be processed by a neural network model to carry out various tasks. Word2vec is one of the popular techniques, and it is described below:

- Word2Vec employs a neural network model with a single hidden layer. It generates real-valued vector representations, called neural word embeddings, for all the words in the vocabulary.
- These word embeddings provide a means to calculate the similarity between words, sentences, and consequently, the documents.
- First step is to create a training corpus as follows:
  - All unique words will make up the vocabulary (W), and each unique word will be given an index between 1 and |W|, where |W| is the number of unique words.
  - Word2vec uses words and their contexts, based on the principle that words with similar context are similar.
    - *Context* of a word is a set of all those words which occur within a window of fixed size in a sentence with the focused word in the middle.
    - Formally, the context of a word with window size fixed to ws includes ws words before and ws words after the focused word, totaling into $2ws$ words.

  - For example, let $w_1, w_2, ..., w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2}, ...w_{n-1}, w_{n-2}$ be a sentence, then the context of word w_i for the window of size 2 is $\{w_{i-2}, w_{i-1}, w_{i+1}, w_{i+2}\}$.
  - The training corpus contains all (word, context) pairs for all the words in the vocabulary, such as for the word w_i: (w_i, w_{i-2}), (w_i, w_{i-1}), (w_i, w_{i+1}), (w_i, w_{i+2}) word-context pairs are in the training set.

- Given the corpus, the word embeddings can be learnt using two different schemes.

  1. *CBOW* (continuous bag of words) takes the context words as input and predicts the probability of the target word corresponding to the input.
  2. *Skip-Gram* works by taking the word as input and predicting the words in its context. So, from each pair in the training corpus, it takes the word as input and predicts the probability distributions corresponding to the context words.

- The learned encodings have some implicit dependencies, unlike one-hot vectors. One hot vector is a binary vector of size |W|, which will have 0s in all positions but 1 in one location that indexes the word being encoded.
- In the following section, we will explain the skip-gram model in more detail because skip-gram gives more precise results than CBOW even while using smaller data sets.

2.8.1 Skip-Gram Model

Figure 2.4 shows the architecture of the skip-gram approach. Some important points to observe from the figure and the working of the skip-gram model are summarized below.

- U_1 is a weight matrix of dimension $|W| \times d$ (where d is the dimension of the word representations) between input and the hidden layer. U_2 is a weight matrix of dimension $d \times |W|$ between hidden and the output layers.
- Multiplication of the input vector with U_1 gives a row of U_1 corresponding to the input. This is the vector of the hidden layer and has a dimension of $1 \times d$. The output vector of dimension $1 \times |W|$ is computed by multiplying this hidden layer vector with U_2. According to the window size (i.e., the number of the context words), this output vector is repeated.
- In the figure, it takes one-hot vector of the word as input and generates two vectors (VECTOR1 and VECTOR2) of probabilities. The ith index of the vector denotes the probability of the ith vocabulary word being the input's context word.

In general, a skip-gram model generates $2.ws$ output vectors, one for each context word, where ws is the window size. After training, the matrix U_1 is the required word representation i.e., each row of the first layer weight matrix is interpreted as the word embedding of the corresponding word in the vocabulary. Thus the goal of word2vec is just to train this weight matrix of the hidden layer and each word will be represented

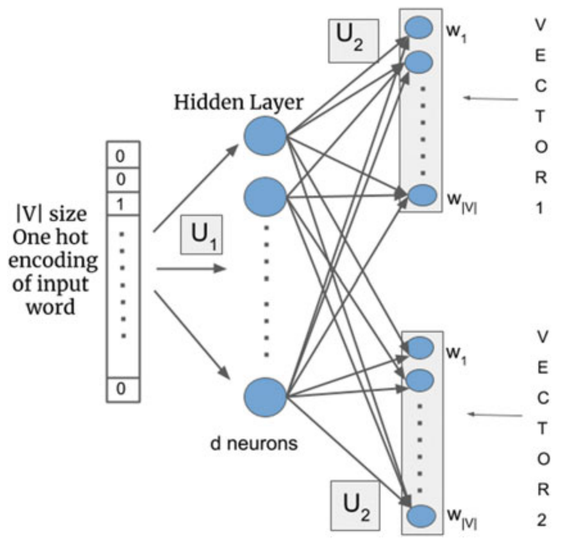

Fig. 2.4 Word2vec: a skip-gram model

by a vector of size $1 \times d$, where d is the dimension and the number of nodes in the hidden layer. The main objective of word2vec is to output similar probability distributions for the words with similar contexts, and consequently, those words should have similar embeddings too. Precisely, the model outputs the probability of being the context word of the input for all the words.

Moreover, when the model is trained on the entire training set, all the weights of the model are modified for each pair in the corpus. So, on a large corpus, the computation becomes challenging to track. Therefore, *negative sampling* is used to train the word2vec model.

- Instead of updating all the weights of the model for each training pair, it will update the parameters corresponding to the positive word along with K negative words.
- Formally saying, for each positive (word, context) pair, it will sample k negative (word, context) pairs and not the complete training set.
- These pairs are called negative because the pair's context word is randomly chosen, not from nearby words.
- Loss is propagated for these selected K entries. The corresponding weights are updated along with the weights of the positive word by maximizing the log-likelihood corresponding to the positive sample and minimizing the log-likelihood of the selected negative pairs.

2.9 Learning Network Embeddings

Any network embedding method generates vectors of low dimension representing an entity of the network such as a node, an edge, or the entire (sub)graph. These are called node embedding, edge embedding, or whole (sub)graph embedding, respectively. Different types of embeddings have diverse applications such as whole graph embedding can facilitate the grouping of multiple graphs together. In contrast, node embeddings are used for node classification, node clustering, and edge prediction tasks.

- Node and Edge Embeddings:
    1. The aim of node embedding is to represent each node of the network as a low-dimensional vector.
        - They preserve relations between the nodes of the network in the form of geometric relations between node embeddings.
        - Each node representation learning approach aims at preserving different properties of the network.
        - So, existing approaches differ in how they encode the similarity between nodes and what node similarity the approach accounts for.
        - For example, some approaches preserve the macroscopic structure that captures scale-free properties. Some maintain first or second-order proximities between nodes, and other methods embed nodes based on their roles in the network.
        - These embeddings are then exploited by many machine learning downstream tasks such as classification and clustering of nodes.
        - Chapter 4 deals with the topic of node embeddings.
    2. Edge embedding aims to encode edges of the network in a vector space.
        - Edges encode node pair relations (pairwise similarity between nodes).
        - Major challenges involve dealing with asymmetry, calculating edge level similarities to encode edge semantics, and facilitating various edge-based tasks such as link prediction.
- Graph Embedding: It aims at representing a set of nodes or an entire graph in a low-dimensional vector space.
    - Embeddings are such that the properties of the entire graph are captured using similarity between graphs, i.e., the algorithm generates a single vector for the whole (sub)graph.
    - An important requirement is to keep similar graphs close in the embedding space.
    - These embeddings have many critical applications, from predicting the class label of an entire graph (graph classification) to clustering graphs.
    - For instance, finding anti-cancer activity, finding molecule toxicity level, and many more can be tackled by embedding the entire graph.

  - Some algorithms first generate embeddings for nodes using node representation learning algorithms and then use any aggregation operator (mean, average, maximum) on those embeddings to output a single vector.
  - A myriad of graph pooling operations is recently developed, which output the graph's coarsened versions and finally represents the entire graph with a single vector.
  - A detailed discussion of these approaches is provided in Chap. 6.

Also, we can group the embedding learning algorithms into Supervised and Unsupervised, as explained below:

- Supervised Learning:
  - Supervised or semi-supervised learning of embeddings depends primarily on label information to learn the model's parameters as the loss is controlled by actual and predicted labels.
  - During the training stage, class label information and information in matrices A (structure) and X (attribute) are exploited. Therefore more discriminative representation learning takes place.
  - Though a small proportion of nodes are labeled, labels proffer information about the categorization of network entities.
  - The learning in this setting depends on the downstream tasks such as classification or link prediction; therefore, these embeddings may be limited to be used for a particular task only.
  - Further, collecting labels is very expensive, which is another disadvantage of these approaches.
- Unsupervised Learning:
  - These approaches can work without the label information while learning representations.
  - So, these methods can be used to learn embeddings even when class label information is not available.
  - Unsupervised learning algorithms use structural information and sometimes attribute information (if available) during the training stage.
  - Since unsupervised algorithms do not exploit any downstream task knowledge for embedding learning; therefore, the embeddings learned are generic and task agnostic, unlike embeddings learned by supervised approaches.
  - An advantage of these embeddings is that they can be used across all downstream tasks for further analysis, such as classification and clustering.

2.10 Summary

- We provided a brief introduction to network representation.
- Graphs are popularly used in representing networks.
- Adjacency matrix associated with the network graph captures the structural properties of the network.
- Content matrix captures the attribute information or content associated with the nodes in the network.
- matrix factorization plays an important role in arriving at a low-dimensional representation of the network entities and even the entire network.
- The low-dimensional embeddings obtained are useful in dealing with several ML tasks including classification, community detection, link prediction, visualization and network construction.
- Word2vec is the most influential tool in network embedding. Random walk over nodes in the graph can be viewed as some kind of sentences.
- High degree nodes will appear in more such walks. The sequences of vertices visited can be viewed like sentences where each node in the walk can be viewed like a word.
- So word2vec can be used in the network/graph applications to embed nodes, where a random walk over vertices is like a sentence and a vertex is like a word in the sentence.
- These ML tasks are instrumental in several important application domains. Some examples are:
  - *E-commerce*: Recommending one or more products to customers
  - *Education*: Finding concepts that are not well understood by a student, interrelationships between different concepts/topics
  - *Health*: Identifying possible drug interactions
  - *Transportation*: Suggesting optimal routes
  - *Agriculture*: Estimating crop yields
  - *Security*: Locating terrorist groups

Bibliography

1. Hamilton WL, Ying R, Leskovec J (2017) Representation learning on graphs: methods and applications. arXiv preprint arXiv:1709.05584
2. Goyal P, Ferrara E (2018) Graph embedding techniques, applications, and performance: a survey. Knowl Based Syst 151:78–94
3. Cui P, Wang X, Pei J, Zhu W (2018) A survey on network embedding. IEEE Trans Knowl Data Eng 31(5):833–852
4. Zhang D, Yin J, Zhu X, Zhang C (2018) Network representation learning: a survey. IEEE Trans Big Data

5. Debnath AK, Lopez de Compadre RL, Debnath G, Shusterman AJ, Hansch C (1991) Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. J Med Chem 34(2):786–797. http://graphkernels.cs.tu-dortmund.de
6. McCallum AK, Nigam K, Rennie J, Seymore K (2000) Automating the construction of internet portals with machine learning. Inf Retr 3(2):127–163
7. Dobson PD, Doig AJ (2003) Distinguishing enzyme structures from non-enzymes without alignments. J Mol Biol 330(4):771–783. http://graphkernels.cs.tu-dortmund.de
8. Schomburg I, Chang A, Ebeling C, Gremse M, Heldt C, Huhn G, Schomburg D (2004) BRENDA, the enzyme database: updates and major new developments. Nucleic Acids Res 32(suppl_1):D431–D433. http://graphkernels.cs.tu-dortmund.de
9. Breitkreutz BJ, Stark C, Reguly T, Boucher L, Breitkreutz A, Livstone M, Oughtred R, Lackner DH, Bähler J, Wood V, Dolinski K (2007) The BioGRID interaction database: 2008 update. Nucleic Acids Res 36(suppl_1):D637–D640
10. Leskovec J, Kleinberg J, Faloutsos C (2007) Graph evolution: densification and shrinking diameters. ACM Trans Knowl Disc Data (TKDD) 1(1):2-es
11. Tang J, Zhang J, Yao L, Li J, Zhang L, Su Z (2008) August. Arnetminer: extraction and mining of academic social networks. In: Proceedings of the 14th ACM SIGKDD international conference on knowledge discovery and data mining, pp 990–998
12. Tang L, Liu H (2009) Relational learning via latent social dimensions. In: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pp 817–826
13. Tang L, Liu H (2009) Scalable learning of collective behavior based on sparse social dimensions. In: Proceedings of the 18th ACM conference on information and knowledge management, pp 1107–1116
14. Kersting K, Kriege NM, Morris C, Mutzel P, Neumann M (2016) Benchmark data sets for graph kernels. http://graphkernels.cs.tu-dortmund.de
15. Lee DD, Seung HS (2001) Algorithms for non-negative matrix factorization. In: Advances in neural information processing systems, pp 556–562
16. Koren Y, Bell R, Volinsky C (2009) Matrix factorization techniques for recommender systems. Computer 42(8):30–37
17. McCormick C (2016) Word2Vec tutorial - the skip-gram model. http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
18. McCormick C (2017) Word2Vec tutorial part 2 - negative sampling. http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/
19. NSS (2017) An intuitive understanding of word embeddings: from count vectors to Word2Vec. https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/