Graph Representation Learning

Notes[1]

Winter 2021

[1] Authors: Eeshaan Jain
Notes of the book Graph Representation Learning by William L. Hamilton

Keyphrases: Graph Representation Learning. Graph Statistics. Graph Kernel Methods. Graph Embeddings. Graph Neural Networks. Graph Generative Models.

This set of notes begin by introducing graphs, and learning tasks on it. Then we discuss the traditional approaches to infer graphs at the node-level and graph-level and further see inter-node relations. This extends to Spectral Graph Theory and use of Graph Laplacians to perform optimal clustering. [Introduction will be updated as file is updated.]

Contents

1 Introduction to Graphs and Learning on Graphs

1.1 Basic definitions

Definition 1.1 (*Graph*)

A graph $G = (V, E)$ is defined by a set of nodes V and a set of edges between these nodes. We denote an edge going from $u \in V$ to $v \in V$ as $(u, v) \in E$.

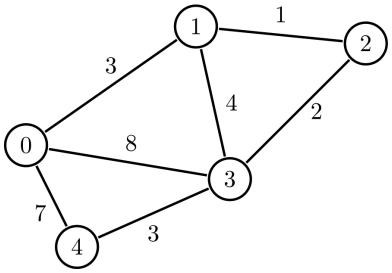

Figure 1: Undirected graph with ordered edges and nodes

Note that definition 1.1 is general, and most of the times we are only concerned with *simple graphs*, which have the following properties:

- For any two nodes in the set V, there exists at most one edge between them.
- There are no self-loops, i.e for any $u \in V$, $(u, u) \notin E$.
- All edges of the graph are *undirected*, i.e for $u, v \in V$, $(u, v) \in E \Leftrightarrow (v, u) \in E$.

Definition 1.2 (*Adjacency Matrix*)

For a simple graph $G = (V, E)$, the adjacency matrix $\mathbf{A} \in \mathbb{R}^{|V| \times |V|}$ is a square matrix with $A_{uv} = 1$ if there is an edge from vertex u to v, and $A_{uv} = 0$ otherwise.

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The adjacency matrix for Fig 1.

Since we are dealing with simple graphs, some immediate consequences are that

- $\mathbf{A}$ is symmetric, i.e $\mathbf{A}^T = \mathbf{A}$.
- $A_{uu} = 0 \; \forall \; u \in V$, i.e all diagonal elements are zero.

Notice that since we are defining the graph in terms of the matrix $\mathbf{A}$, we need to order the nodes prior to the representation (See Fig 1). An extension to definition 1.2 occurs when we allow the use of weighted edges, and in that case for $u, v \in V$, $A_{uv} \in [0, 1]$.

1.2 Multi-relational Graphs

We have noted three types of edges till now: undirected, directed and weighted. But for graphs, it is possible to have different types of edges. In such cases, we extend the notation to include relation type $\tau \in \mathcal{R}$ and write that $(u, \tau, v) \in E$ along with defining an adjacency matrix $\mathbf{A}_\tau$ for each type. The above class of graphs are called multi-relational, and are overall represented by an adjacency tensor $\mathcal{A} \in \mathbb{R}^{|V| \times |\mathcal{R}| \times |V|}$. Two major subsets of such graphs are:

1. **Heterogeneous graphs:** We can partition the set of nodes into k disjoint sets $V = \bigcup_{i=1}^{k} V_i$, and thus we have types imbued on

nodes. Usually in such graphs, we pose constraints on edges, for example they connect nodes of different types only (*multipartite graphs*).

2. **Multiplex graphs:** We can decompose such graphs into a set of layers (see Fig 2), and each node is in every layer with each layer possessing a unique relation. Edges will be present intra-layer and inter-layer.

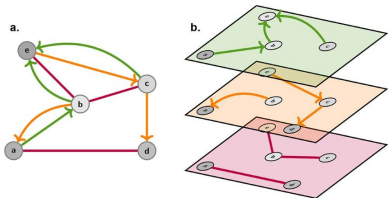

Figure 2: Layered multiplex graph. Source: [Gaudelet et al., 2021]

1.3 *Information representation*

Usually we have features representing graph-level features - may it be on nodes, edges or the entire graph itself. The most common one is node-level, and we associate an $m-$dimensional feature vector with each node, thus overall having a matrix $\mathbf{X} \in \mathbb{R}^{|V| \times m}$.

Now, let's go over some common graphical learning tasks.

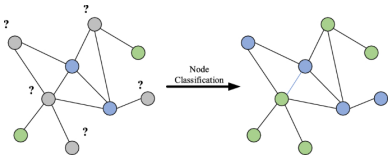

Figure 3: Example of node classification. Source: [Xiao et al., 2022]

1.4 *Node Classification*

The goal of node classification is to predict a label y_u associated with all nodes $u \in V$ when we have been only given the true labels of a subset of the nodes $V_{train} \subset V$. A few examples in literature of the task are:

1. Interactome protein function classification: [Hamilton et al., 2017]
2. Document classification based on citation graphs: [Kipf and Welling, 2016]

Though it seems similar to supervised classification, there is a difficulty that the nodes in a graph are not *i.i.d.* and thus we need to model dependencies between data points. This is exactly what node classification takes a benefit of through

1. **Homophily** [Mcpherson et al., 2001]: Tendency of nodes to share attributes with their neighbors
2. **Structural Equivalence** [Donnat et al., 2017]: Idea that nodes with similar local neighborhood structures will have similar labels
3. **Heterophily**: Presumption that nodes will be preferentially connected to differently-labeled nodes

Note that, the only part of the network we don't access is the labels of the test nodes. We still have access to the structural information, and thus our task is **semi-supervised**.

1.5 *Link Prediction*

This task is related to the inference of edges between nodes in a graph. Given a set of nodes V and an incomplete set of edges $E_{train} \subset E$, we want to infer the set $E \setminus E_{train}$. A few examples in literature of the task are:

1. Content recommendation on social media: [Ying et al., 2018]
2. Modeling side-effects of drugs: [Zitnik et al., 2018]

Although for simple graphs, we can have simple heuristics to give decent results [Lü and Zhou, 2011], the task on multi-relational graphs is much more complex.

Much like node classification, the task boundary is smudged, and it falls under both supervised and unsupervised.

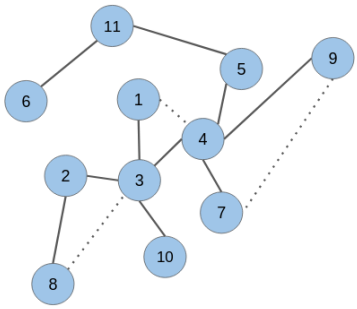

Figure 4: Example of node classification. Source: AnalyticsVidhya

1.6 *Community Detection*

This task is the graphical analog of unsupervised learning. We want to infer latent community structures given only the input graph $G = (V, E)$. A few examples in literature of the task are:

1. Uncovering functional modules in genetic interaction networks: [Agrawal et al., 2017]
2. Fraud detection in financial transaction networks: [Pandit et al., 2007]

1.7 *Other popular graphical tasks*

- ◇ **Graph Regression / Graph Classification**: In these tasks, we want to learn over graph data, and our dataset consists of multiple different graphs and we predict graph-wise. We can treat each graph as i.i.d and thus, the only challenge is to define useful features over graphs.
- ◇ **Graph Clustering** In this task, we learn an unsupervised measure of similarity between pairs of graphs.

2 Traditional Approaches to Graph Learning

2.1 Node-level statistics

Prior to advent of deep learning on graphs, a large part of machine learning on graphs was directed towards extraction of statistics/features based on heuristic functions and domain knowledge from graphs, and passing this information to a machine learning classifier. Now, we look over common node-level statistics.

Definition 2.1 (*Node degree*)

The degree d_u of a node $u \in V$ is the number of edges incident to a node, and in terms of $\mathbf{A}$ for a simple graph

$$d_u = \sum_{v \in V} A_{uv} \tag{1}$$

If our graph is directed or weighted, we can have the notion of in-degree and out-degree by summing over rows or columns in Equation (1).

Perron-Frobenius Theorem: A real square matrix with positive entries has a unique largest real eigenvalue, and the corresponding eigenvector can be chosen to have strictly positive components.

Definition 2.2 (*Eigenvector Centrality*)

The eigenvector centrality e_u of a node $u \in V$ is defined by the recurrence relation

$$e_u = \frac{1}{\lambda} \sum_{v \in V} A_{uv} e_v \; \forall \; u \in V \tag{2}$$

We can rewrite the above equation to notice that $\mathbf{Ae} = \lambda \mathbf{e}$ where $\mathbf{e}$ is the vector of node centralities.

Note that, we want the centrality values to be positive, and by the **Perron-Frobenius Theorem**. In all, $\mathbf{e}$ is given by the eigenvector of the largest eigenvalue of $\mathbf{A}$. Now, since λ is the leading eigenvalue, e_u ranks the likelihood that a node is visited on a random-walk of infinite length on the graph as we can use power iteration method ($\mathbf{e^{(t)}} = \mathbf{Ae^{(t-1)}}$) to find $\mathbf{e}$.

Note that the matrix $\mathbf{B}^{(n)} = \mathbf{A}^n$ has the elements such that $\mathbf{B}^{(n)}_{uv}$ denotes the number of $n-$length paths from $u \to v$.

By centrality, we can also measure how often a node lies on the shortest path between two nodes (*betweeness centrality*), or average shortest path length between a node and all other nodes (*closeness centrality*).

Definition 2.3 (*Clustering Coefficient*)

The clustering coefficient [Watts et al., 2006] is given as

$$c_u = \frac{|(v_1, v_2) \in E \; : \; v_1, v_2 \in \mathcal{N}(u)|}{\binom{d_u}{2}} \tag{3}$$

It essentially measures the proportion of closed triangles in a node's local neighborhood.

We can view clustering coefficient as the ratio between actual number

of triangles and total possible number of triangles within a node's *ego graph*: subgraph having the node, neighbors and all edges between nodes in its neighborhood.

2.2 Graph Kernel Methods

These methods are used for generating graph-level features for tasks such as graph classification.

Definition 2.4 (*Bag of Nodes*)

Bag of nodes refers to aggregates of node-level statistics such as histograms based on clustering coefficients, and the use of such aggregates as graph-level features.

Clearly bag of nodes misses global graph properties, and one way to improve it is to use iterative neighborhood aggregation.

Weisfieler–Lehman Algorithm

1. Assign an initial label $\ell^{(0)}(v)$ to each node. A usual practice is to just assign the degree.
2. Iteratively assign a new label by hashing the multiset of current labels within the neighborhood
$$\ell^{(i)}(v) = \texttt{HASH}(\{\{\ell^{(i-1)}(u) \; \forall \; u \in \mathcal{N}(v)\}\})$$
3. After running K iterations of step-2, our label summarizes a K-hop neighborhood. Now use statistics of labels as features for the graphs.

The WL Kernel is computed by measuring the difference between the label sets of two graphs.

Definition 2.5 (*Graphlets*)

Graphlets are small non-isomorphic induced subgraphs representing connected patterns in a network, and their frequency can be used to assess network structure.

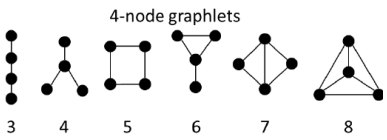

Figure 5: Graphlets with 4 nodes

Graphlet kernels enumerate all graph structures of a possible size and count their occurrence in a graph. This is combinatorially difficult and an alternative is to use path-based methods which examine different kinds of paths in the graph. For example, the *random-walk kernel* [Kashima et al., 2003] involves running random walks all over the graph and then counting the occurrence of different degree sequences, while the *shortest-path kernel* [Borgwardt and Kriegel, 2005] uses shortest paths between nodes instead of random-walks.

2.3 *Measures of Neighborhood Overlap*

Now we try to quantify the relation between nodes, which cannot be done using node-level or graph-level kernels. We denote $\mathbf{S} \in \mathbb{R}^{|V| \times |V|}$ as the similarity matrix summarizing the statistics. We commonly take the probability of an edge occurring proportional to the statistical measure, i.e $\mathbb{P}(A_{uv} = 1) \propto \mathbf{S}[u, v]$.

1. *Local overlap measures*

   (a) **Common Neighbors (CN)**: It is the simplest neighborhood overlap measure and counts the number of neighbors two nodes share.

   $$\mathbf{S}_{CN}[u, v] = |\mathcal{N}(u) \cap \mathcal{N}(v)| \tag{4}$$

   We can extend the idea to local overlap measures defined over CN, denoted by the next three quantifiers.

   (b) **Sorensen index**: This normalizes the counts of common neighbours by the sum of node-degrees.

   $$\mathbf{S}_{Sorensen}[u, v] = \frac{2|\mathcal{N}(u) \cap \mathcal{N}(v)|}{d_u + d_v} \tag{5}$$

   (c) **Salton index**: This normalizes using the product of degrees.

   $$\mathbf{S}_{Salton}[u, v] = \frac{2|\mathcal{N}(u) \cap \mathcal{N}(v)|}{\sqrt{d_u d_v}} \tag{6}$$

   (d) **Jaccard index**: This is an IOU measure.

   $$\mathbf{S}_{Jaccard}[u, v] = \frac{|\mathcal{N}(u) \cap \mathcal{N}(v)|}{|\mathcal{N}(u) \cup \mathcal{N}(v)|} \tag{7}$$

   (e) **Resource Allocation (RA)**: It places importance on the common neighbors by summing the inverse degrees, i.e

   $$\mathbf{S}_{RA}[u, v] = \sum_{w \in \mathcal{N}(u) \cap \mathcal{N}(v)} \frac{1}{d_w} \tag{8}$$

   (f) **Adamic-Adar (AA)**: It is similar to RA, just utilizes inverse log of the degrees, i.e

   $$\mathbf{S}_{AA}[u, v] = \sum_{w \in \mathcal{N}(u) \cap \mathcal{N}(v)} \frac{1}{\log d_w} \tag{9}$$

   Both these measures infer that common neighbors with low degrees are more informative.

2. *Global overlap measures*

(a) **Katz Index**: We take a discounted count of number of paths of all lengths between a pair of nodes, i.e

$$\mathbf{S}_{Katz}[u,v] = \sum_{i=1}^{\infty} \beta^i A^i_{uv} = \Big((\mathbf{I} - \beta\mathbf{A})^{-1} - \mathbf{I}\Big)[u,v] \tag{10}$$

Here $\beta \in \mathbb{R}^+$ is our discount, and a small $\beta < 1$ down-weights the importance of long-paths.

(b) **Leicht, Holme, and Newman Similarity**: The issue with Katz index is that it is biased towards high-degree nodes. To avoid that, we normalize the number of paths by its expectation, i.e $\frac{\mathbf{A}^i}{\mathbb{E}[\mathbf{A}^i]}$. The expectations can be computed by drawing a random graph with the same set of degrees as the graph under test, and obtain

$$\mathbb{E}[A_{uv}] = \frac{d_u d_v}{2|E|} \qquad \mathbb{E}[A^2_{uv}] = \frac{d_u d_v}{(2|E|)^2} \sum_{w \in V} d_w(d_w - 1) \tag{11}$$

Calculation for $i > 2$ needs to be approximated, and to do so we utilize the fact mentioned earlier (2.1) to notice that the growth is by a factor λ for large i, and thus $\mathbb{E}[A^i_{uv}] = \frac{d_u d_v \lambda^{i-1}}{2|E|}$. Finally, the index is given as

$$\mathbf{S}_{LHN}[u,v] = I_{uv} + \frac{2|E|}{d_u d_v} \sum_{i=0}^{\infty} \beta\lambda^{i-1} A^i_{uv} \tag{12}$$

(c) **Random Walk Methods**: Take $\mathbf{D}$ to be a diagonal matrix with entries as the node degrees. Looking at the Personalized PageRank algorithm [Rajaraman and Ullman, 2012], we define $\mathbf{P} = AD^{-1}$ and define vector $\mathbf{q}_u$ as the probability vector of visiting all nodes starting at node u. In a recursive fashion we incorporate the probability of restarting the random walk with probability c and write $\mathbf{q}_u = c\mathbf{P}\mathbf{q}_u + (1-c)\mathbf{e}_u$ where $\mathbf{e}_u$ is a one-hot vector at node u. It's solution is

$$\mathbf{q}_u = (1-c)(1-c\mathbf{P})^{-1}\mathbf{e}_u \tag{13}$$

The similarity measure is

$$\mathbf{S}_{RW}[u,v] = \mathbf{q}_u[v] + \mathbf{q}_v[u] \tag{14}$$

2.4 *Graph Laplacians*

Laplacians are transformations of the adjacency matrix which have useful properties.

Definition 2.6 (*Unnormalized Laplacian*)

The unnormalized Laplacian is given by $\mathbf{L} = \mathbf{D} - \mathbf{A}$ *and for simple*

graphs has the following properties:

- ◇ *It is symmetric and positive semi-definite.*
- ◇ *For* $\mathbf{x} \in \mathbb{R}^{|V|}$,

$$\mathbf{x}^T\mathbf{L}\mathbf{x} = \sum_{(u,v)\in E} (\mathbf{x}[u] - \mathbf{x}[v])^2 \tag{15}$$

- ◇ *The eigenvalues of* $\mathbf{L}$ *are non-negative.*
- ◇ *The geometric multiplicity of* $\mathbf{L}$ *denotes the number of connected components in the graphs.*

We extend the definition to normalized Laplacians in two ways:

$$\mathbf{L}_{sym} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2} \qquad \mathbf{L}_{RW} = \mathbf{D}^{-1}\mathbf{L} \tag{16}$$

2.5 Clustering using Laplacians

We can extend property 4 of Laplacians to give optimal clustering within a fully-connected graph.

Definition 2.7 (*Graph Cut*)

Let $U \subset V$ *and let* $\bar{U}$ *be its complement. Given K non-overlapping partitions of a graph* $\{U_i\}_{i=1}^K$*, we define the cut value as*

$$cut(U_1, \cdots, U_K) = \frac{1}{2}\sum_{k=1}^{K} |(u,v) \in E \ : \ u \in U_k, v \in \bar{U}_k| \tag{17}$$

It essentially counts the number of edges crossing the boundary between partitions.

We simply can't minimize this as it yields single-node clusters. Hence, we normalize it in two ways:

$$RatioCut(U_1, \cdots, U_K) = \frac{1}{2}\sum_{k=1}^{K} \frac{|(u,v) \in E \ : \ u \in U_k, v \in \bar{U}_k|}{|U_k|} \tag{18}$$

$$NCut(U_1, \cdots, U_K) = \frac{1}{2}\sum_{k=1}^{K} \frac{|(u,v) \in E \ : \ u \in U_k, v \in \bar{U}_k|}{\sum_{w\in U_k} d_w} \tag{19}$$

The former penalizes small cluster sizes, while the latter forces similar number of edges incident on nodes in all clusters. Minimizing them can be defined as optimal clustering. The optimal solution for 2-clustering is derived here 2.5. The above derivation can be done for $NCut$ but the result is has ν_2 for $\mathbf{L}_{RW}$.

2-clustering over RatioCut: We solve

$$\min_{U\in V} RatioCut(U, \bar{U})$$

Define $\mathbf{a} \in \mathbb{R}^{|V|}$ and $\xi = \sqrt{\frac{|\bar{U}|}{|U|}}$ such that $a_u = \xi$ if $u \in U$ else $a_u = -\frac{1}{\xi}$. With some calculation, we can show that

$$\mathbf{a}^T\mathbf{L}\mathbf{a} = |V| RatioCut(U, \bar{U})$$

We can also show that

1. $\sum_{u\in V} a_u = 0 \Leftrightarrow \mathbf{a} \perp \mathbf{1}$
2. $||a||^2 = |V|$

Solving $\min_{U\in V} \mathbf{a}^T\mathbf{L}\mathbf{a}$ and the given constraints is NP-hard as our set is discrete. We remove discreteness and solve

$$\min_{U\in\mathbb{R}^{|V|}} \mathbf{a}^T\mathbf{L}\mathbf{a}$$

using the **Rayleigh-Ritz Theorem** to get that $\mathbf{a} = \nu_2$, i.e the second smallest eigenvector of $\mathbf{L}$ (note that the smallest eigenvector is just $\mathbf{1}$). Finally, we discretize the nodes based on the sign of a_u, i.e $u \in U$ if $a_u \geq 0$ else $u \in \bar{U}$.

(*Note:* Smallest eigenvector denotes the eigenvector for the smallest eigenvalue).

2.6 Spectral Clustering

We can use the principle of spectral clustering by representing the nodes as a spectrum of $\mathbf{L}$, and do an approximate optimal clustering

as follows:

General Spectral Clustering Algorithm

1. Find K smallest eigenvectors of $\mathbf{L}$ except the smallest.
2. Let $\mathbf{U} \in \mathbb{R}^{|V| \times (K-1)}$ and have columns as the eigenvectors from above.
3. Let $\mathbf{z}_u = \mathbf{U}[u]$ be the row for node u.
4. Run K-means clustering on the embeddings $\mathbf{z}_u \, \forall u \in V$.

2.7 *Learning Representations*

Clearly, although the traditional approaches are useful many times, they are hand-engineered and thus are inflexible, i.e cannot adapt to changes. Also, designing them is time-consuming and expensive. Thus, we would benefit if we could learn structural representations over graphs.

References

[Agrawal et al., 2017] Agrawal, M., Zitnik, M., and Leskovec, J. (2017). Large-scale analysis of disease pathways in the human interactome.

[Borgwardt and Kriegel, 2005] Borgwardt, K. and Kriegel, H. (2005). Shortest-path kernels on graphs. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 8 pp.–.

[Donnat et al., 2017] Donnat, C., Zitnik, M., Hallac, D., and Leskovec, J. (2017). Spectral graph wavelets for structural role similarity in networks.

[Gaudelet et al., 2021] Gaudelet, T., Day, B., Jamasb, A., Soman, J., Regep, C., Liu, G., Hayter, J., Vickers, R., Roberts, C., Tang, J., Roblin, D., Blundell, T., Bronstein, M., and Taylor-King, J. (2021). Utilizing graph machine learning within drug discovery and development. *Briefings in Bioinformatics*, 22.

[Hamilton et al., 2017] Hamilton, W. L., Ying, R., and Leskovec, J. (2017). Inductive representation learning on large graphs. *CoRR*, abs/1706.02216.

[Kashima et al., 2003] Kashima, H., Tsuda, K., and Inokuchi, A. (2003). Marginalized kernels between labeled graphs. volume 1, pages 321–328.

[Kipf and Welling, 2016] Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907.

[Lü and Zhou, 2011] Lü, L. and Zhou, T. (2011). Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170.

[Mcpherson et al., 2001] Mcpherson, M., Smith-Lovin, L., and Cook, J. (2001). Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 27:415–.

[Pandit et al., 2007] Pandit, S., Chau, D. H., Wang, S., and Faloutsos, C. (2007). Netprobe: A fast and scalable system for fraud detection in online auction networks.

[Rajaraman and Ullman, 2012] Rajaraman, A. and Ullman, J. (2012). *Mining of Massive Datasets*. Cambridge University Press.

[Watts et al., 2006] Watts, D., J, D., Strogatz, and H, S. (2006). *Collective dynamics of 'small world' networks*, pages 301–303.

[Xiao et al., 2022] Xiao, S., Wang, S., Dai, Y., and Guo, W. (2022). Graph neural networks in node classification: survey and evaluation. *Machine Vision and Applications*, 33.

[Ying et al., 2018] Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. (2018). Graph convolutional neural networks for web-scale recommender systems. *CoRR*, abs/1806.01973.

[Zitnik et al., 2018] Zitnik, M., Agrawal, M., and Leskovec, J. (2018). Modeling polypharmacy side effects with graph convolutional networks.