Chapter 2

Background and Traditional Approaches

Before we introduce the concepts of graph representation learning and deep learning on graphs, it is necessary to give some methodological background and context. What kinds of methods were used for machine learning on graphs prior to the advent of modern deep learning approaches? In this chapter, we will provide a very brief and focused tour of traditional learning approaches over graphs, providing pointers and references to more thorough treatments of these methodological approaches along the way. This background chapter will also serve to introduce key concepts from graph analysis that will form the foundation for later chapters.

Our tour will be roughly aligned with the different kinds of learning tasks on graphs. We will begin with a discussion of basic graph statistics, kernel methods, and their use for node and graph classification tasks. Following this, we will introduce and discuss various approaches for measuring the overlap between node neighborhoods, which form the basis of strong heuristics for relation prediction. Finally, we will close this background section with a brief introduction of spectral clustering using graph Laplacians. Spectral clustering is one of the most well-studied algorithms for clustering or community detection on graphs, and our discussion of this technique will also introduce key mathematical concepts that will re-occur throughout this book.

2.1 Graph Statistics and Kernel Methods

Traditional approaches to classification using graph data follow the standard machine learning paradigm that was popular prior to the advent of deep learning. We begin by extracting some statistics or features—based on heuristic functions or domain knowledge—and then use these features as input to a standard machine learning classifier (e.g., logistic regression). In this section, we will first introduce some important node-level features and statistics, and we will fol-

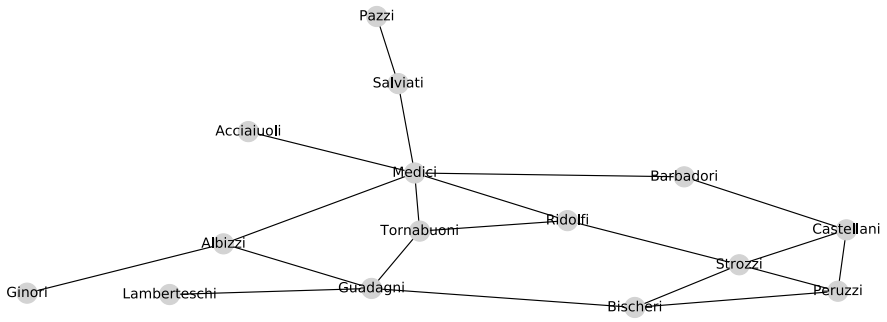

Figure 2.1: A visualization of the marriages between various different prominent families in 15th century Florence [Padgett and Ansell, 1993].

low this by a discussion of how these node-level statistics can be generalized to graph-level statistics and extended to design kernel methods over graphs. Our goal will be to introduce various heuristic statistics and graph properties, which are often used as features in traditional machine learning pipelines applied to graphs.

2.1.1 Node-level statistics and features

Following Jackson [2010], we will motivate our discussion of node-level statistics and features with a simple (but famous) social network: the network of 15th century Florentine marriages (Figure 2.1). This social network is well-known due to the work of Padgett and Ansell [1993], which used this network to illustrate the rise in power of the Medici family (depicted near the center) who came to dominate Florentine politics. Political marriages were an important way to consolidate power during the era of the Medicis, so this network of marriage connections encodes a great deal about the political structure of this time.

For our purposes, we will consider this network and the rise of the Medici from a machine learning perspective and ask the question: What features or statistics could a machine learning model use to predict the Medici's rise? In other words, what properties or statistics of the Medici node distinguish it from the rest of the graph? And, more generally, what are useful properties and statistics that we can use to characterize the nodes in this graph?

In principle the properties and statistics we discuss below could be used as features in a node classification model (e.g., as input to a logistic regression model). Of course, we would not be able to realistically train a machine learning model on a graph as small as the Florentine marriage network. However, it is still illustrative to consider the kinds of features that could be used to differentiate the nodes in such a real-world network, and the properties we discuss are generally useful across a wide variety of node classification tasks.

Node degree. The most obvious and straightforward node feature to examine is *degree*, which is usually denoted d_u for a node $u \in \mathcal{V}$ and simply counts the number of edges incident to a node:

$$d_u = \sum_{v \in V} \mathbf{A}[u, v]. \tag{2.1}$$

Note that in cases of directed and weighted graphs, one can differentiate between different notions of degree—e.g., corresponding to outgoing edges or incoming edges by summing over rows or columns in Equation (2.1). In general, the degree of a node is an essential statistic to consider, and it is often one of the most informative features in traditional machine learning models applied to node-level tasks.

In the case of our illustrative Florentine marriages graph, we can see that degree is indeed a good feature to distinguish the Medici family, as they have the highest degree in the graph. However, their degree only outmatches the two closest families—the Strozzi and the Guadagni—by a ratio of 3 to 2. Are there perhaps additional or more discriminative features that can help to distinguish the Medici family from the rest of the graph?

Node centrality

Node degree simply measures how many neighbors a node has, but this is not necessarily sufficient to measure the *importance* of a node in a graph. In many cases—such as our example graph of Florentine marriages—we can benefit from additional and more powerful measures of node importance. To obtain a more powerful measure of importance, we can consider various measures of what is known as node *centrality*, which can form useful features in a wide variety of node classification tasks.

One popular and important measure of centrality is the so-called *eigenvector centrality*. Whereas degree simply measures how many neighbors each node has, eigenvector centrality also takes into account how important a node's neighbors are. In particular, we define a node's eigenvector centrality e_u via a recurrence relation in which the node's centrality is proportional to the average centrality of its neighbors:

$$e_u = \frac{1}{\lambda} \sum_{v \in V} \mathbf{A}[u, v] e_v \; \forall u \in \mathcal{V}, \tag{2.2}$$

where λ is a constant. Rewriting this equation in vector notation with $\mathbf{e}$ as the vector of node centralities, we can see that this recurrence defines the standard eigenvector equation for the adjacency matrix:

$$\lambda \mathbf{e} = \mathbf{A}\mathbf{e}. \tag{2.3}$$

In other words, the centrality measure that satisfies the recurrence in Equation 2.2 corresponds to an eigenvector of the adjacency matrix. Assuming that

we require positive centrality values, we can apply the Perron-Frobenius Theorem[1] to further determine that the vector of centrality values $\mathbf{e}$ is given by the eigenvector corresponding to the largest eigenvalue of $\mathbf{A}$ [Newman, 2016].

One view of eigenvector centrality is that it ranks the likelihood that a node is visited on a random walk of infinite length on the graph. This view can be illustrated by considering the use of power iteration to obtain the eigenvector centrality values. That is, since λ is the leading eigenvector of $\mathbf{A}$, we can compute $\mathbf{e}$ using power iteration via[2]

$$\mathbf{e}^{(t+1)} = \mathbf{A}\mathbf{e}^{(t)}. \tag{2.4}$$

If we start off this power iteration with the vector $\mathbf{e}^{(0)} = (1, 1, ..., 1)^\top$, then we can see that after the first iteration $\mathbf{e}^{(1)}$ will contain the degrees of all the nodes. In general, at iteration $t \geq 1$, $\mathbf{e}^{(t)}$ will contain the number of length-t paths arriving at each node. Thus, by iterating this process indefinitely we obtain a score that is proportional to the number of times a node is visited on paths of infinite length. This connection between node importance, random walks, and the spectrum of the graph adjacency matrix will return often throughout the ensuing sections and chapters of this book.

Returning to our example of the Florentine marriage network, if we compute the eigenvector centrality values on this graph, we again see that the Medici family is the most influential, with a normalized value of 0.43 compared to the next-highest value of 0.36. There are, of course, other measures of centrality that we could use to characterize the nodes in this graph—some of which are even more discerning with respect to the Medici family's influence. These include *betweeness centrality*—which measures how often a node lies on the shortest path between two other nodes—as well as *closeness centrality*—which measures the average shortest path length between a node and all other nodes. These measures and many more are reviewed in detail by Newman [2018].

The clustering coefficient

Measures of importance, such as degree and centrality, are clearly useful for distinguishing the prominent Medici family from the rest of the Florentine marriage network. But what about features that are useful for distinguishing between the other nodes in the graph? For example, the Peruzzi and Guadagni nodes in the graph have very similar degree (3 v.s. 4) and similar eigenvector centralities (0.28 v.s. 0.29). However, looking at the graph in Figure 2.1, there is a clear difference between these two families. Whereas the the Peruzzi family is in the midst of a relatively tight-knit cluster of families, the Guadagni family occurs in a more "star-like" role.

[1] The Perron-Frobenius Theorem is a fundamental result in linear algebra, proved independently by Oskar Perron and Georg Frobenius [Meyer, 2000]. The full theorem has many implications, but for our purposes the key assertion in the theorem is that any irreducible square matrix has a unique largest real eigenvalue, which is the only eigenvalue whose corresponding eigenvector can be chosen to have strictly positive components.

[2] Note that we have ignored the normalization in the power iteration computation for simplicity, as this does not change the main result.

This important structural distinction can be measured using variations of the *clustering coefficient*, which measures the proportion of closed triangles in a node's local neighborhood. The popular *local variant* of the clustering coefficient is computed as follows [Watts and Strogatz, 1998]:

$$c_u = \frac{|(v_1, v_2) \in \mathcal{E} \ : \ v_1, v_2 \in \mathcal{N}(u)|}{\binom{d_u}{2}}. \tag{2.5}$$

The numerator in this equation counts the number of edges between neighbours of node u (where we use $\mathcal{N}(u) = \{v \in \mathcal{V} \ : \ (u, v) \in \mathcal{E}\}$ to denote the node neighborhood). The denominator calculates how many pairs of nodes there are in u's neighborhood.

The clustering coefficient takes its name from the fact that it measures how tightly clustered a node's neighborhood is. A clustering coefficient of 1 would imply that all of u's neighbors are also neighbors of each other. In our Florentine marriage graph, we can see that some nodes are highly clustered—e.g., the Peruzzi node has a clustering coefficient of 0.66—while other nodes such as the Guadagni node have clustering coefficients of 0. As with centrality, there are numerous variations of the clustering coefficient (e.g., to account for directed graphs), which are also reviewed in detail by Newman [2018]. An interesting and important property of real-world networks throughout the social and biological sciences is that they tend to have far higher clustering coefficients than one would expect if edges were sampled randomly [Watts and Strogatz, 1998].

Closed triangles, ego graphs, and motifs.

An alternative way of viewing the clustering coefficient—rather than as a measure of local clustering—is that it counts the number of closed triangles within each node's local neighborhood. In more precise terms, the clustering coefficient is related to the ratio between the actual number of triangles and the total possible number of triangles within a node's *ego graph*, i.e., the subgraph containing that node, its neighbors, and all the edges between nodes in its neighborhood.

This idea can be generalized to the notion of counting arbitrary *motifs* or *graphlets* within a node's ego graph. That is, rather than just counting triangles, we could consider more complex structures, such as cycles of particular length, and we could characterize nodes by counts of how often these different motifs occur in their ego graph. Indeed, by examining a node's ego graph in this way, we can essentially transform the task of computing node-level statistics and features to a graph-level task. Thus, we will now turn our attention to this graph-level problem.

2.1.2 Graph-level features and graph kernels

So far we have discussed various statistics and properties at the node level, which could be used as features for node-level classification tasks. However, what if our goal is to do graph-level classification? For example, suppose we are given a dataset of graphs representing molecules and our goal is to classify the

solubility of these molecules based on their graph structure. How would we do this? In this section, we will briefly survey approaches to extracting graph-level features for such tasks.

Many of the methods we survey here fall under the general classification of *graph kernel methods*, which are approaches to designing features for graphs or implicit kernel functions that can be used in machine learning models. We will touch upon only a small fraction of the approaches within this large area, and we will focus on methods that extract explicit feature representations, rather than approaches that define implicit kernels (i.e., similarity measures) between graphs. We point the interested reader to Kriege et al. [2020] and Vishwanathan et al. [2010] for detailed surveys of this area.

Bag of nodes

The simplest approach to defining a graph-level feature is to just aggregate node-level statistics. For example, one can compute histograms or other summary statistics based on the degrees, centralities, and clustering coefficients of the nodes in the graph. This aggregated information can then be used as a graph-level representation. The downside to this approach is that it is entirely based upon local node-level information and can miss important global properties in the graph.

The Weisfeiler-Lehman kernel

One way to improve the basic bag of nodes approach is using a strategy of *iterative neighborhood aggregation.* The idea with these approaches is to extract node-level features that contain more information than just their local ego graph, and then to aggregate these richer features into a graph-level representation.

Perhaps the most important and well-known of these strategies is the Weisfeiler-Lehman (WL) algorithm and kernel [Shervashidze et al., 2011, Weisfeiler and Lehman, 1968]. The basic idea behind the WL algorithm is the following:

1. First, we assign an initial label $l^{(0)}(v)$ to each node. In most graphs, this label is simply the degree, i.e., $l^{(0)}(v) = d_v \; \forall v \in V$.

2. Next, we iteratively assign a new label to each node by hashing the multi-set of the current labels within the node's neighborhood:

$$l^{(i)}(v) = \text{HASH}(\{\{l^{(i-1)}(u) \; \forall u \in \mathcal{N}(v)\}\}), \tag{2.6}$$

   where the double-braces are used to denote a multi-set and the HASH function maps each unique multi-set to a unique new label.

3. After running K iterations of re-labeling (i.e., Step 2), we now have a label $l^{(K)}(v)$ for each node that summarizes the structure of its K-hop neighborhood. We can then compute histograms or other summary statistics over these labels as a feature representation for the graph. In other words, the WL kernel is computed by measuring the difference between the resultant label sets for two graphs.

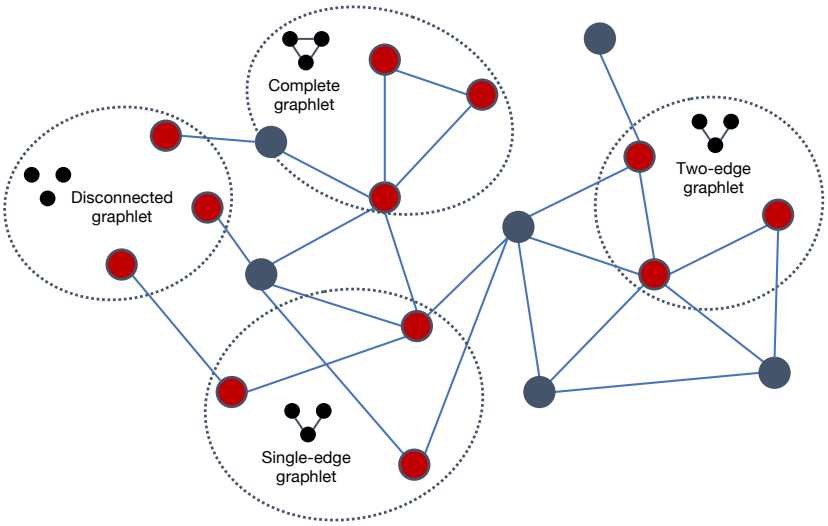

Figure 2.2: The four different size-3 graphlets that can occur in a simple graph.

The WL kernel is popular, well-studied and known to have important theoretical properties. For example, one popular way to approximate graph isomorphism is to check whether or not two graphs have the same label set after K rounds of the WL algorithm, and this approach is known to solve the isomorphism problem for a broad set of graphs [Shervashidze et al., 2011].

Graphlets and path-based methods

Finally, just as in our discussion of node-level features, one valid and powerful strategy for defining features over graphs is to simply count the occurrence of different small subgraph structures, usually called *graphlets* in this context. Formally, the graphlet kernel involves enumerating all possible graph structures of a particular size and counting how many times they occur in the full graph. (Figure 2.2 illustrates the various graphlets of size 3). The challenge with this approach is that counting these graphlets is a combinatorially difficult problem, though numerous approximations have been proposed [Shervashidze and Borgwardt, 2009].

An alternative to enumerating all possible graphlets is to use *path-based methods*. In these approaches, rather than enumerating graphlets, one simply examines the different kinds of *paths* that occur in the graph. For example, the random walk kernel proposed by Kashima et al. [2003] involves running random walks over the graph and then counting the occurrence of different degree sequences,[3] while the shortest-path kernel of Borgwardt and Kriegel [2005] involves a similar idea but uses only the shortest-paths between nodes (rather

[3]Other node labels can also be used.

than random walks). As we will see in Chapter 3 of this book, this idea of characterizing graphs based on walks and paths is a powerful one, as it can extract rich structural information while avoiding many of the combinatorial pitfalls of graph data.

2.2 Neighborhood Overlap Detection

In the last section we covered various approaches to extract features or statistics about individual nodes or entire graphs. These node and graph-level statistics are useful for many classification tasks. However, they are limited in that they do not quantify the *relationships* between nodes. For instance, the statistics discussed in the last section are not very useful for the task of relation prediction, where our goal is to predict the existence of an edge between two nodes (Figure 2.3).

In this section we will consider various statistical measures of neighborhood overlap between pairs of nodes, which quantify the extent to which a pair of nodes are related. For example, the simplest neighborhood overlap measure just counts the number of neighbors that two nodes share:

$$\mathbf{S}[u, v] = |\mathcal{N}(u) \cap \mathcal{N}(v)|, \tag{2.7}$$

where we use $\mathbf{S}[u, v]$ to denote the value quantifying the relationship between nodes u and v and let $\mathbf{S} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ denote the *similarity matrix* summarizing all the pairwise node statistics.

Even though there is no "machine learning" involved in any of the statistical measures discussed in this section, they are still very useful and powerful baselines for relation prediction. Given a neighborhood overlap statistic $\mathbf{S}[u, v]$, a common strategy is to assume that the likelihood of an edge (u, v) is simply proportional to $\mathbf{S}[u, v]$:

$$P(\mathbf{A}[u, v] = 1) \propto \mathbf{S}[u, v]. \tag{2.8}$$

Thus, in order to approach the relation prediction task using a neighborhood overlap measure, one simply needs to set a threshold to determine when to predict the existence of an edge. Note that in the relation prediction setting we generally assume that we only know a subset of the true edges $\mathcal{E}_{\text{train}} \subset \mathcal{E}$. Our hope is that node-node similarity measures computed on the training edges will lead to accurate predictions about the existence of test (i.e., unseen) edges (Figure 2.3).

2.2.1 Local overlap measures

Local overlap statistics are simply functions of the number of common neighbors two nodes share, i.e. $|\mathcal{N}(u) \cap \mathcal{N}(v)|$. For instance, the Sorensen index defines a matrix $\mathbf{S}_{\text{Sorenson}} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ of node-node neighborhood overlaps with entries given by

$$\mathbf{S}_{\text{Sorenson}}[u, v] = \frac{2|\mathcal{N}(u) \cap \mathcal{N}(v)|}{d_u + d_v}, \tag{2.9}$$

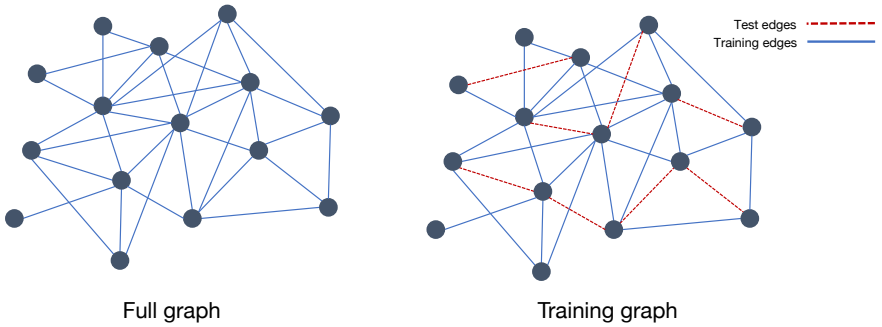

Figure 2.3: An illustration of a full graph and a subsampled graph used for training. The dotted edges in the training graph are removed when training a model or computing the overlap statistics. The model is evaluated based on its ability to predict the existence of these held-out *test edges*.

which normalizes the count of common neighbors by the sum of the node degrees. Normalization of some kind is usually very important; otherwise, the overlap measure would be highly biased towards predicting edges for nodes with large degrees. Other similar approaches include the the Salton index, which normalizes by the product of the degrees of u and v, i.e.

$$\mathbf{S}_{\text{Salton}}[u, v] = \frac{2|\mathcal{N}(u) \cap \mathcal{N}(v)|}{\sqrt{d_u d_v}}, \tag{2.10}$$

as well as the Jaccard overlap:

$$\mathbf{S}_{\text{Jaccard}}[u, v] = \frac{|\mathcal{N}(u) \cap \mathcal{N}(v)|}{|\mathcal{N}(u) \cup \mathcal{N}(v)|}. \tag{2.11}$$

In general, these measures seek to quantify the overlap between node neighborhoods while minimizing any biases due to node degrees. There are many further variations of this approach in the literature [Lü and Zhou, 2011].

There are also measures that go beyond simply counting the number of common neighbors and that seek to consider the *importance* of common neighbors in some way. The Resource Allocation (RA) index counts the inverse degrees of the common neighbors,

$$\mathbf{S}_{\text{RA}}[v_1, v_2] = \sum_{u \in \mathcal{N}(v_1) \cap \mathcal{N}(v_2)} \frac{1}{d_u}, \tag{2.12}$$

while the Adamic-Adar (AA) index performs a similar computation using the inverse logarithm of the degrees:

$$\mathbf{S}_{\text{AA}}[v_1, v_2] = \sum_{u \in \mathcal{N}(v_1) \cap \mathcal{N}(v_2)} \frac{1}{\log(d_u)}. \tag{2.13}$$

Both these measures give more weight to common neighbors that have low degree, with intuition that a shared low-degree neighbor is more informative than a shared high-degree one.

2.2.2 Global overlap measures

Local overlap measures are extremely effective heuristics for link prediction and often achieve competitive performance even compared to advanced deep learning approaches [Perozzi et al., 2014]. However, the local approaches are limited in that they only consider local node neighborhoods. For example, two nodes could have no local overlap in their neighborhoods but still be members of the same community in the graph. *Global overlap* statistics attempt to take such relationships into account.

Katz index

The Katz index is the most basic global overlap statistic. To compute the Katz index we simply count the number of paths *of all lengths* between a pair of nodes:

$$\mathbf{S}_{\text{Katz}}[u, v] = \sum_{i=1}^{\infty} \beta^i \mathbf{A}^i[u, v], \tag{2.14}$$

where $\beta \in \mathbb{R}^+$ is a user-defined parameter controlling how much weight is given to short versus long paths. A small value of $\beta < 1$ would down-weight the importance of long paths.

Geometric series of matrices The Katz index is one example of a geometric series of matrices, variants of which occur frequently in graph analysis and graph representation learning. The solution to a basic geometric series of matrices is given by the following theorem:

Theorem 1. *Let* $\mathbf{X}$ *be a real-valued square matrix and let* λ_1 *denote the largest eigenvalue of* $\mathbf{X}$*. Then*

$$(\mathbf{I} - \mathbf{X})^{-1} = \sum_{i=0}^{\infty} \mathbf{X}^i$$

if and only if $\lambda_1 < 1$ *and* $(\mathbf{I} - \mathbf{X})$ *is non-singular.*

Proof. Let $s_n = \sum_{i=0}^{n} \mathbf{X}^i$ then we have that

$$\begin{aligned} \mathbf{X}s_n &= \mathbf{X}\sum_{i=0}^{n} \mathbf{X}^i \\ &= \sum_{i=1}^{n+1} \mathbf{X}^i \end{aligned}$$

and

$$\begin{aligned} s_n - \mathbf{X}s_n &= \sum_{i=0}^{n} \mathbf{X}^i - \sum_{i=1}^{n+1} \mathbf{X}^i \\ s_n(\mathbf{I} - \mathbf{X}) &= \mathbf{I} - \mathbf{X}^{n+1} \\ s_n &= (\mathbf{I} - \mathbf{X}^{n+1})(\mathbf{I} - \mathbf{X})^{-1} \end{aligned}$$

And if $\lambda_1 < 1$ we have that $\lim_{n\to\infty} \mathbf{X}^n = 0$ so

$$\begin{aligned} \lim_{n\to\infty} s_n &= \lim_{n\to\infty} (\mathbf{I} - \mathbf{X}^{n+1})(\mathbf{I} - \mathbf{X})^{-1} \\ &= \mathbf{I}(\mathbf{I} - \mathbf{X})^{-1} \\ &= (\mathbf{I} - \mathbf{X})^{-1} \end{aligned}$$

□

Based on Theorem 1, we can see that the solution to the Katz index is given by

$$\mathbf{S}_{\text{Katz}} = (\mathbf{I} - \beta\mathbf{A})^{-1} - \mathbf{I}, \tag{2.15}$$

where $\mathbf{S}_{\text{Katz}} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ is the full matrix of node-node similarity values.

Leicht, Holme, and Newman (LHN) similarity

One issue with the Katz index is that it is strongly biased by node degree. Equation (2.14) is generally going to give higher overall similarity scores when considering high-degree nodes, compared to low-degree ones, since high-degree nodes will generally be involved in more paths. To alleviate this, Leicht et al. [2006] propose an improved metric by considering the ratio between the actual number of observed paths and the *number of expected paths between two nodes*:

$$\frac{\mathbf{A}^i}{\mathbb{E}[\mathbf{A}^i]}, \tag{2.16}$$

i.e., the number of paths between two nodes is normalized based on our expectation of how many paths we expect under a random model.

To compute the expectation $\mathbb{E}[\mathbf{A}^i]$, we rely on what is called the *configuration model*, which assumes that we draw a random graph with the same set of degrees as our given graph. Under this assumption, we can analytically compute that

$$\mathbb{E}[\mathbf{A}[u, v]] = \frac{d_u d_v}{2m}, \tag{2.17}$$

where we have used $m = |\mathcal{E}|$ to denote the total number of edges in the graph. Equation (2.17) states that under a random configuration model, the likelihood of an edge is simply proportional to the product of the two node degrees. This

can be seen by noting that there are d_u edges leaving u and each of these edges has a $\frac{d_v}{2m}$ chance of ending at v. For $\mathbb{E}[\mathbf{A}^2[u,v]]$ we can similarly compute

$$\mathbb{E}[\mathbf{A}^2[v_1,v_2]] = \frac{d_{v_1}d_{v_2}}{(2m)^2}\sum_{u\in\mathcal{V}}(d_u-1)d_u. \tag{2.18}$$

This follows from the fact that path of length 2 could pass through any intermediate vertex u, and the likelihood of such a path is proportional to the likelihood that an edge leaving v_1 hits u—given by $\frac{d_{v_1}d_u}{2m}$—multiplied by the probability that an edge leaving u hits v_2—given by $\frac{d_{v_2}(d_u-1)}{2m}$ (where we subtract one since we have already used up one of u's edges for the incoming edge from v_1).

Unfortunately the analytical computation of expected node path counts under a random configuration model becomes intractable as we go beyond paths of length three. Thus, to obtain the expectation $\mathbb{E}[\mathbf{A}^i]$ for longer path lengths (i.e., $i > 2$), Leicht et al. [2006] rely on the fact the largest eigenvalue can be used to approximate the growth in the number of paths. In particular, if we define $\mathbf{p}_i \in \mathbb{R}^{|\mathcal{V}|}$ as the vector counting the number of length-i paths between node u and all other nodes, then we have that for large i

$$\mathbf{A}\mathbf{p}_i = \lambda_1\mathbf{p}_{i-1}, \tag{2.19}$$

since $\mathbf{p}_i$ will eventually converge to the dominant eigenvector of the graph. This implies that the number of paths between two nodes grows by a factor of λ_1 at each iteration, where we recall that λ_1 is the largest eigenvalue of $\mathbf{A}$. Based on this approximation for large i as well as the exact solution for $i = 1$ we obtain:

$$\mathbb{E}[\mathbf{A}^i[u,v]] = \frac{d_u d_v \lambda^{i-1}}{2m}. \tag{2.20}$$

Finally, putting it all together we can obtain a normalized version of the Katz index, which we term the LNH index (based on the initials of the authors who proposed the algorithm):

$$\mathbf{S}_{\text{LNH}}[u,v] = \mathbf{I}[u,v] + \frac{2m}{d_u d_v}\sum_{i=0}^{\infty}\beta^i\lambda_1^{1-i}\mathbf{A}^i[u,v], \tag{2.21}$$

where $\mathbf{I}$ is a $|\mathcal{V}| \times |\mathcal{V}|$ identity matrix indexed in a consistent manner as $\mathbf{A}$. Unlike the Katz index, the LNH index accounts for the *expected* number of paths between nodes and only gives a high similarity measure if two nodes occur on more paths than we expect. Using Theorem 1 the solution to the matrix series (after ignoring diagonal terms) can be written as [Lü and Zhou, 2011]:

$$\mathbf{S}_{\text{LNH}} = 2\alpha m\lambda_1\mathbf{D}^{-1}(\mathbf{I} - \frac{\beta}{\lambda_1}\mathbf{A})^{-1}\mathbf{D}^{-1}, \tag{2.22}$$

where $\mathbf{D}$ is a matrix with node degrees on the diagonal.

Random walk methods

Another set of global similarity measures consider random walks rather than exact counts of paths over the graph. For example, we can directly apply a variant of the famous PageRank approach [Page et al., 1999][4]—known as the Personalized PageRank algorithm [Leskovec et al., 2020]—where we define the stochastic matrix $\mathbf{P} = \mathbf{A}\mathbf{D}^{-1}$ and compute:

$$\mathbf{q}_u = c\mathbf{P}\mathbf{q}_u + (1-c)\mathbf{e}_u. \tag{2.23}$$

In this equation $\mathbf{e}_u$ is a one-hot indicator vector for node u and $\mathbf{q}_u[v]$ gives the stationary probability that random walk starting at node u visits node v. Here, the c term determines the probability that the random walk restarts at node u at each timestep. Without this restart probability, the random walk probabilities would simply converge to a normalized variant of the eigenvector centrality. However, with this restart probability we instead obtain a measure of importance specific to the node u, since the random walks are continually being "teleported" back to that node. The solution to this recurrence is given by

$$\mathbf{q}_u = (1-c)(\mathbf{I} - c\mathbf{P})^{-1}\mathbf{e}_u, \tag{2.24}$$

and we can define a node-node random walk similarity measure as

$$\mathbf{S}_{\text{RW}}[u, v] = \mathbf{q}_u[v] + \mathbf{q}_v[u], \tag{2.25}$$

i.e., the similarity between a pair of nodes is proportional to how likely we are to reach each node from a random walk starting from the other node.

2.3 Graph Laplacians and Spectral Methods

Having discussed traditional approaches to classification with graph data (Section 2.1) as well as traditional approaches to relation prediction (Section 2.2), we now turn to the problem of learning to cluster the nodes in a graph. This section will also motivate the task of learning low dimensional embeddings of nodes. We begin with the definition of some important matrices that can be used to represent graphs and a brief introduction to the foundations of *spectral graph theory*.

2.3.1 Graph Laplacians

Adjacency matrices can represent graphs without any loss of information. However, there are alternative matrix representations of graphs that have useful mathematical properties. These matrix representations are called *Laplacians* and are formed by various transformations of the adjacency matrix.

[4]PageRank was developed by the founders of Google and powered early versions of the search engine.

Unnormalized Laplacian

The most basic Laplacian matrix is the unnormalized Laplacian, defined as follows:

$$\mathbf{L} = \mathbf{D} - \mathbf{A}, \tag{2.26}$$

where $\mathbf{A}$ is the adjacency matrix and $\mathbf{D}$ is the degree matrix. The Laplacian matrix of a simple graph has a number of important properties:

1. It is symmetric ($\mathbf{L}^T = \mathbf{L}$) and positive semi-definite ($\mathbf{x}^\top \mathbf{L}\mathbf{x} \geq 0, \forall \mathbf{x} \in \mathbb{R}^{|\mathcal{V}|}$).

2. The following vector identity holds $\forall \mathbf{x} \in \mathbb{R}^{|\mathcal{V}|}$

$$\mathbf{x}^\top \mathbf{L}\mathbf{x} = \frac{1}{2} \sum_{u \in \mathcal{V}} \sum_{v \in \mathcal{V}} \mathbf{A}[u, v](\mathbf{x}[u] - \mathbf{x}[v])^2 \tag{2.27}$$

$$= \sum_{(u,v) \in \mathcal{E}} (\mathbf{x}[u] - \mathbf{x}[v])^2 \tag{2.28}$$

3. $\mathbf{L}$ has $|V|$ non-negative eigenvalues: $0 = \lambda_{|\mathcal{V}|} \leq \lambda_{|\mathcal{V}|-1} \leq ... \leq \lambda_1$

The Laplacian and connected components The Laplacian summarizes many important properties of the graph. For example, we have the following theorem:

Theorem 2. *The geometric multiplicity of the* 0 *eigenvalue of the Laplacian* $\mathbf{L}$ *corresponds to the number of connected components in the graph.*

Proof. This can be seen by noting that for any eigenvector $\mathbf{e}$ of the eigenvalue 0 we have that

$$\mathbf{e}^\top \mathbf{L}\mathbf{e} = 0 \tag{2.29}$$

by the definition of the eigenvalue-eigenvector equation. And, the result in Equation (2.29) implies that

$$\sum_{(u,v) \in \mathcal{E}} (\mathbf{e}[u] - \mathbf{e}[v])^2 = 0. \tag{2.30}$$

The equality above then implies that $\mathbf{e}[u] = \mathbf{e}[v], \forall (u, v) \in \mathcal{E}$, which in turn implies that $\mathbf{e}[u]$ is the same constant for all nodes u that are in the same connected component. Thus, if the graph is fully connected then the eigenvector for the eigenvalue 0 will be a constant vector of ones for all nodes in the graph, and this will be the only eigenvector for eigenvalue 0, since in this case there is only one unique solution to Equation (2.29).

Conversely, if the graph is composed of multiple connected components then we will have that Equation 2.29 holds independently on each of the blocks of the Laplacian corresponding to each connected component. That is, if the graph is composed of K connected components, then there exists

an ordering of the nodes in the graph such that the Laplacian matrix can be written as

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_1 & & & \\ & \mathbf{L}_2 & & \\ & & \ddots & \\ & & & \mathbf{L}_K \end{bmatrix}, \tag{2.31}$$

where each of the $\mathbf{L}_k$ blocks in this matrix is a valid graph Laplacian of a fully connected subgraph of the original graph. Since they are valid Laplacians of fully connected graphs, for each of the $\mathbf{L}_k$ blocks we will have that Equation (2.29) holds and that each of these sub-Laplacians has an eigenvalue of 0 with multiplicity 1 and an eigenvector of all ones (defined only over the nodes in that component). Moreover, since $\mathbf{L}$ is a block diagonal matrix, its spectrum is given by the union of the spectra of all the $\mathbf{L}_k$ blocks, i.e., the eigenvalues of $\mathbf{L}$ are the union of the eigenvalues of the $\mathbf{L}_k$ matrices and the eigenvectors of $\mathbf{L}$ are the union of the eigenvectors of all the $\mathbf{L}_k$ matrices with 0 values filled at the positions of the other blocks. Thus, we can see that each block contributes one eigenvector for eigenvalue 0, and this *eigenvector is an indicator vector for the nodes in that connected component.* □

Normalized Laplacians

In addition to the unnormalized Laplacian there are also two popular normalized variants of the Laplacian. The symmetric normalized Laplacian is defined as

$$\mathbf{L}_{\text{sym}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}, \tag{2.32}$$

while the random walk Laplacian is defined as

$$\mathbf{L}_{\text{RW}} = \mathbf{D}^{-1}\mathbf{L} \tag{2.33}$$

Both of these matrices have similar properties as the Laplacian, but their algebraic properties differ by small constants due to the normalization. For example, Theorem 2 holds exactly for $\mathbf{L}_{\text{RW}}$. For $\mathbf{L}_{\text{sym}}$, Theorem 2 holds but with the eigenvectors for the 0 eigenvalue scaled by $\mathbf{D}^{\frac{1}{2}}$. As we will see throughout this book, these different variants of the Laplacian can be useful for different analysis and learning tasks.

2.3.2 Graph Cuts and Clustering

In Theorem 2, we saw that the eigenvectors corresponding to the 0 eigenvalue of the Laplacian can be used to assign nodes to clusters based on which connected component they belong to. However, this approach only allows us to cluster

nodes that are already in disconnected components, which is trivial. In this section, we take this idea one step further and show that the Laplacian can be used to give an optimal clustering of nodes *within a fully connected graph.*

Graph cuts

In order to motivate the Laplacian spectral clustering approach, we first must define what we mean by an *optimal* cluster. To do so, we define the notion of a *cut* on a graph. Let $\mathcal{A} \subset \mathcal{V}$ denote a subset of the nodes in the graph and let $\bar{\mathcal{A}}$ denote the complement of this set, i.e., $\mathcal{A} \cup \bar{\mathcal{A}} = \mathcal{V}, \mathcal{A} \cap \bar{\mathcal{A}} = \emptyset$. Given a partitioning of the graph into K non-overlapping subsets $\mathcal{A}_1, ..., \mathcal{A}_K$ we define the cut value of this partition as

$$\text{cut}(\mathcal{A}_1, ..., \mathcal{A}_K) = \frac{1}{2} \sum_{k=1}^{K} |(u, v) \in \mathcal{E} \ : \ u \in \mathcal{A}_k, v \in \bar{\mathcal{A}}_k|. \tag{2.34}$$

In other words, the cut is simply the count of how many edges cross the boundary between the partition of nodes. Now, one option to define an *optimal clustering* of the nodes into K clusters would be to select a partition that minimizes this cut value. There are efficient algorithms to solve this task, but a known problem with this approach is that it tends to simply make clusters that consist of a single node [Stoer and Wagner, 1997].

Thus, instead of simply minimizing the cut we generally seek to minimize the cut while also enforcing that the partitions are all reasonably large. One popular way of enforcing this is by minimizing the *Ratio Cut*:

$$\text{RatioCut}(\mathcal{A}_1, ..., \mathcal{A}_K) = \frac{1}{2} \sum_{k=1}^{K} \frac{|(u, v) \in \mathcal{E} \ : \ u \in \mathcal{A}_k, v \in \bar{\mathcal{A}}_k|}{|\mathcal{A}_k|}, \tag{2.35}$$

which penalizes the solution for choosing small cluster sizes. Another popular solution is to minimize the *Normalized Cut (NCut)*:

$$\text{NCut}(\mathcal{A}_1, ..., \mathcal{A}_K) = \frac{1}{2} \sum_{k=1}^{K} \frac{|(u, v) \in \mathcal{E} \ : \ u \in \mathcal{A}_k, v \in \bar{\mathcal{A}}_k|}{\text{vol}(\mathcal{A}_k)}, \tag{2.36}$$

where $\text{vol}(\mathcal{A}) = \sum_{u \in \mathcal{A}} d_u$. The NCut enforces that all clusters have a similar number of edges incident to their nodes.

Approximating the RatioCut with the Laplacian spectrum

We will now derive an approach to find a cluster assignment that minimizes the RatioCut using the Laplacian spectrum. (A similar approach can be used to minimize the NCut value as well.) For simplicity, we will consider the case where we $K = 2$ and we are separating our nodes into two clusters. Our goal is to solve the following optimization problem

$$\min_{\mathcal{A} \in \mathcal{V}} \text{RatioCut}(\mathcal{A}, \bar{\mathcal{A}}). \tag{2.37}$$

To rewrite this problem in a more convenient way, we define the following vector $\mathbf{a} \in \mathbb{R}^{|\mathcal{V}|}$:

$$\mathbf{a}[u] = \begin{cases} \sqrt{\frac{|\bar{\mathcal{A}}|}{|\mathcal{A}|}} & \text{if } u \in \mathcal{A} \\ -\sqrt{\frac{|\mathcal{A}|}{|\bar{\mathcal{A}}|}} & \text{if } u \in \bar{\mathcal{A}} \end{cases}. \tag{2.38}$$

Combining this vector with our properties of the graph Laplacian we can see that

$$\begin{aligned} \mathbf{a}^\top \mathbf{L}\mathbf{a} &= \sum_{(u,v)\in\mathcal{E}} (\mathbf{a}[u] - \mathbf{a}[v])^2 && (2.39)\\ &= \sum_{(u,v)\in\mathcal{E}\,:\,u\in\mathcal{A},v\in\bar{\mathcal{A}}} (\mathbf{a}[u] - \mathbf{a}[v])^2 && (2.40)\\ &= \sum_{(u,v)\in\mathcal{E}\,:\,u\in\mathcal{A},v\in\bar{\mathcal{A}}} \left(\sqrt{\frac{|\bar{\mathcal{A}}|}{|\mathcal{A}|}} - \left(-\sqrt{\frac{|\mathcal{A}|}{|\bar{\mathcal{A}}|}}\right)\right)^2 && (2.41)\\ &= \text{cut}(\mathcal{A},\bar{\mathcal{A}}) \left(\frac{|\mathcal{A}|}{|\bar{\mathcal{A}}|} + \frac{|\bar{\mathcal{A}}|}{|\mathcal{A}|} + 2\right) && (2.42)\\ &= \text{cut}(\mathcal{A},\bar{\mathcal{A}}) \left(\frac{|\mathcal{A}| + |\bar{\mathcal{A}}|}{|\bar{\mathcal{A}}|} + \frac{|\mathcal{A}| + |\bar{\mathcal{A}}|}{|\mathcal{A}|}\right) && (2.43)\\ &= |\mathcal{V}|\text{RatioCut}(\mathcal{A},\bar{\mathcal{A}}). && (2.44) \end{aligned}$$

Thus, we can see that $\mathbf{a}$ allows us to write the Ratio Cut in terms of the Laplacian (up to a constant factor). In addition, $\mathbf{a}$ has two other important properties:

$$\sum_{u\in\mathcal{V}} \mathbf{a}[u] = 0, \text{ or equivalently, } \mathbf{a} \perp \mathbb{1} \qquad \text{(Property 1)} \tag{2.45}$$

$$\|\mathbf{a}\|^2 = |\mathcal{V}| \qquad \text{(Property 2)}, \tag{2.46}$$

where $\mathbb{1}$ is the vector of all ones.

Putting this all together we can rewrite the Ratio Cut minimization problem in Equation (2.37) as

$$\begin{aligned} &\min_{\mathcal{A}\subset\mathcal{V}} \mathbf{a}^\top \mathbf{L}\mathbf{a} && (2.47)\\ &\quad s.t. \\ &\qquad \mathbf{a} \perp \mathbb{1} \\ &\qquad \|\mathbf{a}\|^2 = |\mathcal{V}| \\ &\qquad \mathbf{a} \text{ defined as in Equation 2.38.} \end{aligned}$$

Unfortunately, however, this is an NP-hard problem since the restriction that $\mathbf{a}$ is defined as in Equation 2.38 requires that we are optimizing over a discrete set. The obvious relaxation is to remove this discreteness condition and simplify

the minimization to be over real-valued vectors:

$$\begin{aligned} \min_{\mathbf{a}\in\mathbb{R}^{|\mathcal{V}|}} \ & \mathbf{a}^\top \mathbf{L}\mathbf{a} \\ s.t. \ & \\ & \mathbf{a} \perp \mathbb{1} \\ & \|\mathbf{a}\|^2 = |\mathcal{V}|. \end{aligned} \tag{2.48}$$

By the Rayleigh-Ritz Theorem, the solution to this optimization problem is given by the second-smallest eigenvector of $\mathbf{L}$ (since the smallest eigenvector is equal to $\mathbb{1}$).

Thus, we can approximate the minimization of the RatioCut by setting $\mathbf{a}$ to be the second-smallest eigenvector[5] of the Laplacian. To turn this real-valued vector into a set of discrete cluster assignments, we can simply assign nodes to clusters based on the sign of $\mathbf{a}[u]$, i.e.,

$$\begin{cases} u \in \mathcal{A} & \text{if } \mathbf{a}[u] \geq 0 \\ u \in \bar{\mathcal{A}} & \text{if } \mathbf{a}[u] < 0. \end{cases} \tag{2.49}$$

In summary, the second-smallest eigenvector of the Laplacian is a continuous approximation to the discrete vector that gives an optimal cluster assignment (with respect to the RatioCut). An analogous result can be shown for approximating the NCut value, but it relies on the second-smallest eigenvector of the normalized Laplacian $\mathbf{L}_{\text{RW}}$ [Von Luxburg, 2007].

2.3.3 Generalized spectral clustering

In the last section we saw that the spectrum of the Laplacian allowed us to find a meaningful partition of the graph into two clusters. In particular, we saw that the second-smallest eigenvector could be used to partition the nodes into different clusters. This general idea can be extended to an arbitrary number of K clusters by examining the K smallest eigenvectors of the Laplacian. The steps of this general approach are as follows:

1. Find the K smallest eigenvectors of $\mathbf{L}$ (excluding the smallest): $\mathbf{e}_{|\mathcal{V}|-1}, \mathbf{e}_{|\mathcal{V}|-2}, ..., \mathbf{e}_{|\mathcal{V}|-K}$.
2. Form the matrix $\mathbf{U} \in \mathbb{R}^{|\mathcal{V}|\times(K-1)}$ with the eigenvectors from Step 1 as columns.
3. Represent each node by its corresponding row in the matrix $\mathbf{U}$, i.e.,
$$\mathbf{z}_u = \mathbf{U}[u] \; \forall u \in \mathcal{V}.$$
4. Run K-means clustering on the *embeddings* $\mathbf{z}_u \; \forall u \in \mathcal{V}$.

[5]Note that by second-smallest eigenvector we mean the eigenvector corresponding to the second-smallest eigenvalue.

As with the discussion of the $K = 2$ case in the previous section, this approach can be adapted to use the normalized Laplacian , and the approximation result for $K = 2$ can also be generalized to this $K > 2$ case [Von Luxburg, 2007].

The general principle of spectral clustering is a powerful one. We can represent the nodes in a graph using the spectrum of the graph Laplacian, and this representation can be motivated as a principled approximation to an optimal graph clustering. There are also close theoretical connections between spectral clustering and random walks on graphs, as well as the field of graph signal processing Ortega et al. [2018]. We will discuss many of these connections in future chapters.

2.4 Towards Learned Representations

In the previous sections, we saw a number of traditional approaches to learning over graphs. We discussed how graph statistics and kernels can extract feature information for classification tasks. We saw how neighborhood overlap statistics can provide powerful heuristics for relation prediction. And, we offered a brief introduction to the notion of spectral clustering, which allows us to cluster nodes into communities in a principled manner. However, the approaches discussed in this chapter—and especially the node and graph-level statistics—are limited due to the fact that they require careful, hand-engineered statistics and measures. These hand-engineered features are inflexible—i.e., they cannot adapt through a learning process—and designing these features can be a time-consuming and expensive process. The following chapters in this book introduce alternative approach to learning over graphs: *graph representation learning.* Instead of extracting hand-engineered features, we will seek to *learn* representations that encode structural information about the graph.