Chapter 1

Introduction

Graphs are a ubiquitous data structure and a universal language for describing complex systems. In the most general view, a graph is simply a collection of objects (i.e., nodes), along with a set of interactions (i.e., edges) between pairs of these objects. For example, to encode a social network as a graph we might use nodes to represent individuals and use edges to represent that two individuals are friends (Figure 1.1). In the biological domain we could use the nodes in a graph to represent proteins, and use the edges to represent various biological interactions, such as kinetic interactions between proteins.

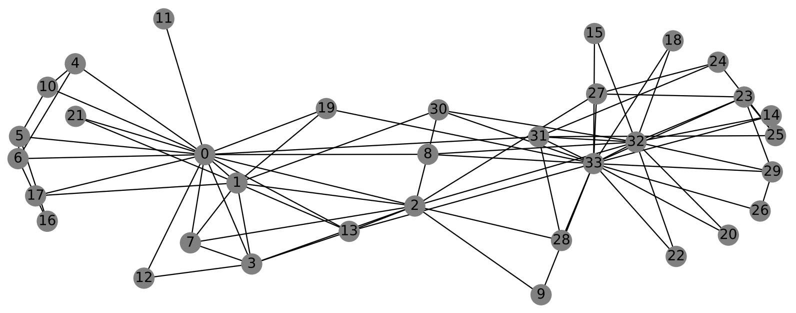

Figure 1.1: The famous *Zachary Karate Club Network* represents the friendship relationships between members of a karate club studied by Wayne W. Zachary from 1970 to 1972. An edge connects two individuals if they socialized outside of the club. During Zachary's study, the club split into two factions—centered around nodes 0 and 33—and Zachary was able to correctly predict which nodes would fall into each faction based on the graph structure [Zachary, 1977].

The power of the graph formalism lies both in its focus on *relationships between points* (rather than the properties of individual points), as well as in its generality. The same graph formalism can be used to represent social networks, interactions between drugs and proteins, the interactions between atoms in a molecule, or the connections between terminals in a telecommunications network—to name just a few examples.

Graphs do more than just provide an elegant theoretical framework, however. They offer a mathematical foundation that we can build upon to analyze, understand, and learn from real-world complex systems. In the last twenty-five years, there has been a dramatic increase in the quantity and quality of graph-structured data that is available to researchers. With the advent of large-scale social networking platforms, massive scientific initiatives to model the interactome, food webs, databases of molecule graph structures, and billions of interconnected web-enabled devices, there is no shortage of meaningful graph data for researchers to analyze. The challenge is unlocking the potential of this data.

This book is about how we can use *machine learning* to tackle this challenge. Of course, machine learning is not the only possible way to analyze graph data.[1] However, given the ever-increasing scale and complexity of the graph datasets that we seek to analyze, it is clear that machine learning will play an important role in advancing our ability to model, analyze, and understand graph data.

1.1 What is a graph?

Before we discuss machine learning on graphs, it is necessary to give a bit more formal description of what exactly we mean by "graph data". Formally, a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined by a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$ between these nodes. We denote an edge going from node $u \in \mathcal{V}$ to node $v \in \mathcal{V}$ as $(u, v) \in \mathcal{E}$. In many cases we will be concerned only with *simple graphs*, where there is at most one edge between each pair of nodes, no edges between a node and itself, and where the edges are all undirected, i.e., $(u, v) \in \mathcal{E} \leftrightarrow (v, u) \in \mathcal{E}$.

A convenient way to represent graphs is through an *adjacency matrix* $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$. To represent a graph with an adjacency matrix, we order the nodes in the graph so that every node indexes a particular row and column in the adjacency matrix. We can then represent the presence of edges as entries in this matrix: $\mathbf{A}[u, v] = 1$ if $(u, v) \in \mathcal{E}$ and $\mathbf{A}[u, v] = 0$ otherwise. If the graph contains only undirected edges then $\mathbf{A}$ will be a symmetric matrix, but if the graph is *directed* (i.e., edge direction matters) then $\mathbf{A}$ will not necessarily be symmetric. Some graphs can also have *weighted* edges, where the entries in the adjacency matrix are arbitrary real-values rather than $\{0, 1\}$. For instance, a weighted edge in a protein-protein interaction graph might indicated the strength of the association between two proteins.

1.1.1 Multi-relational Graphs

Beyond the distinction between undirected, directed and weighted edges, we will also consider graphs that have different *types* of edges. For instance, in graphs representing drug-drug interactions, we might want different edges to

[1]The field of *network analysis* independent of machine learning is the subject of entire textbooks and will not be covered in detail here [Newman, 2018].

correspond to different side effects that can occur when you take a pair of drugs at the same time. In these cases we can extend the edge notation to include an edge or relation type τ, e.g., $(u, \tau, v) \in \mathcal{E}$, and we can define one adjacency matrix $\mathbf{A}_\tau$ per edge type. We call such graphs *multi-relational*, and the entire graph can be summarized by an adjacency tensor $\mathcal{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{R}|\times|\mathcal{V}|}$, where $\mathcal{R}$ is the set of relations. Two important subsets of multi-relational graphs are often known as *heterogeneous* and *multiplex* graphs.

Heterogeneous graphs In heterogeneous graphs, nodes are also imbued with *types*, meaning that we can partition the set of nodes into disjoint sets $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2 \cup ... \cup \mathcal{V}_k$ where $\mathcal{V}_i \cap \mathcal{V}_j = \emptyset, \forall i \neq j$. Edges in heterogeneous graphs generally satisfy constraints according to the node types, most commonly the constraint that certain edges only connect nodes of certain types, i.e., $(u, \tau_i, v) \in \mathcal{E} \to u \in \mathcal{V}_j, v \in \mathcal{V}_k$. For example, in a heterogeneous biomedical graph, there might be one type of node representing proteins, one type of representing drugs, and one type representing diseases. Edges representing "treatments" would only occur between drug nodes and disease nodes. Similarly, edges representing "polypharmacy side-effects" would only occur between two drug nodes. *Multipartite* graphs are a well-known special case of heterogeneous graphs, where edges can only connect nodes that have different types, i.e., $(u, \tau_i, v) \in \mathcal{E} \to u \in \mathcal{V}_j, v \in \mathcal{V}_k \wedge j \neq k$.

Multiplex graphs In multiplex graphs we assume that the graph can be decomposed in a set of *k layers*. Every node is assumed to belong to every layer, and each layer corresponds to a unique relation, representing the *intra-layer* edge type for that layer. We also assume that *inter-layer* edges types can exist, which connect the same node across layers. Multiplex graphs are best understood via examples. For instance, in a multiplex transportation network, each node might represent a city and each layer might represent a different mode of transportation (e.g., air travel or train travel). Intra-layer edges would then represent cities that are connected by different modes of transportation, while inter-layer edges represent the possibility of switching modes of transportation within a particular city.

1.1.2 Feature Information

Lastly, in many cases we also have *attribute* or *feature* information associated with a graph (e.g., a profile picture associated with a user in a social network). Most often these are node-level attributes that we represent using a real-valued matrix $\mathbf{X} \in \mathbb{R}^{|V|\times m}$, where we assume that the ordering of the nodes is consistent with the ordering in the adjacency matrix. In heterogeneous graphs we generally assume that each different type of node has its own distinct type of attributes. In rare cases we will also consider graphs that have real-valued edge features in addition to discrete edge types, and in some cases we even associate real-valued features with entire graphs.

Graph or network? We use the term "graph" in this book, but you will see many other resources use the term "network" to describe the same kind of data. In some places, we will use both terms (e.g., for social or biological networks). So which term is correct? In many ways, this terminological difference is a historical and cultural one: the term "graph" appears to be more prevalent in machine learning community[a], but "network" has historically been popular in the data mining and (unsurprisingly) network science communities. We use both terms in this book, but we also make a distinction between the usage of these terms. We use the term *graph* to describe the abstract data structure that is the focus of this book, but we will also often use the term *network* to describe specific, real-world instantiations of this data structure (e.g., social networks). This terminological distinction is fitting with their current popular usages of these terms. *Network analysis* is generally concerned with the properties of real-world data, whereas *graph theory* is concerned with the theoretical properties of the mathematical graph abstraction.

[a]Perhaps in some part due to the terminological clash with "neural networks."

1.2 Machine learning on graphs

Machine learning is inherently a problem-driven discipline. We seek to build models that can learn from data in order to solve particular tasks, and machine learning models are often categorized according to the type of task they seek to solve: Is it a *supervised* task, where the goal is to predict a target output given an input datapoint? Is it an *unsupervised* task, where the goal is to infer patterns, such as clusters of points, in the data?

Machine learning with graphs is no different, but the usual categories of supervised and unsupervised are not necessarily the most informative or useful when it comes to graphs. In this section we provide a brief overview of the most important and well-studied machine learning tasks on graph data. As we will see, "supervised" problems are popular with graph data, but machine learning problems on graphs often blur the boundaries between the traditional machine learning categories.

1.2.1 Node classification

Suppose we are given a large social network dataset with millions of users, but we know that a significant number of these users are actually bots. Identifying these bots could be important for many reasons: a company might not want to advertise to bots or bots may actually be in violation of the social network's terms of service. Manually examining every user to determine if they are a bot would be prohibitively expensive, so ideally we would like to have a model that could classify users as a bot (or not) given only a small number of manually labeled examples.

This is a classic example of *node classification*, where the goal is to predict the label y_u—which could be a type, category, or attribute—associated with all the nodes $u \in \mathcal{V}$, when we are only given the true labels on a *training set* of nodes $\mathcal{V}_{\text{train}} \subset \mathcal{V}$. Node classification is perhaps the most popular machine learning task on graph data, especially in recent years. Examples of node classification beyond social networks include classifying the function of proteins in the interactome [Hamilton et al., 2017b] and classifying the topic of documents based on hyperlink or citation graphs [Kipf and Welling, 2016a]. Often, we assume that we have label information only for a very small subset of the nodes in a single graph (e.g., classifying bots in a social network from a small set of manually labeled examples). However, there are also instances of node classification that involve many labeled nodes and/or that require generalization across disconnected graphs (e.g., classifying the function of proteins in the interactomes of different species).

At first glance, node classification appears to be a straightforward variation of standard supervised classification, but there are in fact important differences. The most important difference is that the nodes in a graph are not *independent and identically distributed (i.i.d.)*. Usually, when we build supervised machine learning models we assume that each datapoint is statistically *independent* from all the other datapoints; otherwise, we might need to model the dependencies between all our input points. We also assume that the datapoints are *identically distributed*; otherwise, we have no way of guaranteeing that our model will generalize to new datapoints. Node classification completely breaks this i.i.d. assumption. Rather than modeling a set of i.i.d. datapoints, we are instead modeling an interconnected set of nodes.

In fact, the key insight behind many of the most successful node classification approaches is to explicitly leverage the connections between nodes. One particularly popular idea is to exploit *homophily*, which is the tendency for nodes to share attributes with their neighbors in the graph [McPherson et al., 2001]. For example, people tend to form friendships with others who share the same interests or demographics. Based on the notion of homophily we can build machine learning models that try to assign similar labels to neighboring nodes in a graph [Zhou et al., 2004]. Beyond homophily there are also concepts such as *structural equivalence* [Donnat et al., 2018], which is the idea that nodes with similar local neighborhood structures will have similar labels, as well as *heterophily*, which presumes that nodes will be preferentially connected to nodes with different labels.[2] When we build node classification models we want to exploit these concepts and model the relationships between nodes, rather than simply treating nodes as independent datapoints.

Supervised or semi-supervised? Due to the atypical nature of node classification, researchers often refer to it as *semi-supervised* [Yang et al., 2016]. This terminology is used because when we are training node classi-

[2]For example, gender is an attribute that exhibits heterophily in many social networks.

fication models, we usually have access to the full graph, including all the unlabeled (e.g., test) nodes. The only thing we are missing is the labels of test nodes. However, we can still use information about the test nodes (e.g., knowledge of their neighborhood in the graph) to improve our model during training. This is different from the usual supervised setting, in which unlabeled datapoints are completely unobserved during training.

The general term used for models that combine labeled and unlabeled data during traning is semi-supervised learning, so it is understandable that this term is often used in reference to node classification tasks. It is important to note, however, that standard formulations of semi-supervised learning still require the i.i.d. assumption, which does not hold for node classification. Machine learning tasks on graphs do not easily fit our standard categories!

1.2.2 Relation prediction

Node classification is useful for inferring information about a node based on its relationship with other nodes in the graph. But what about cases where we are missing this relationship information? What if we know only some of protein-protein interactions that are present in a given cell, but we want to make a good guess about the interactions we are missing? Can we use machine learning to infer the edges between nodes in a graph?

This task goes by many names, such as link prediction, graph completion, and relational inference, depending on the specific application domain. We will simply call it *relation prediction* here. Along with node classification, it is one of the more popular machine learning tasks with graph data and has countless real-world applications: recommending content to users in social platforms [Ying et al., 2018a], predicting drug side-effects [Zitnik et al., 2018], or inferring new facts in a relational databases [Bordes et al., 2013]—all of these tasks can be viewed as special cases of relation prediction.

The standard setup for relation prediction is that we are given a set of nodes $\mathcal{V}$ and an incomplete set of edges between these nodes $\mathcal{E}_{\text{train}} \subset \mathcal{E}$. Our goal is to use this partial information to infer the missing edges $\mathcal{E} \setminus \mathcal{E}_{\text{train}}$. The complexity of this task is highly dependent on the type of graph data we are examining. For instance, in simple graphs, such as social networks that only encode "friendship" relations, there are simple heuristics based on how many neighbors two nodes share that can achieve strong performance [Lü and Zhou, 2011]. On the other hand, in more complex multi-relational graph datasets, such as biomedical knowledge graphs that encode hundreds of different biological interactions, relation prediction can require complex reasoning and inference strategies [Nickel et al., 2016]. Like node classification, relation prediction blurs the boundaries of traditional machine learning categories—often being referred to as both supervised and unsupervised—and it requires inductive biases that are specific to the graph domain. In addition, like node classification, there are many variants of relation prediction, including settings where the predictions are made over a single, fixed graph [Lü and Zhou, 2011], as well as settings where relations must be predicted across multiple disjoint graphs [Teru et al., 2020].

1.2.3 Clustering and community detection

Both node classification and relation prediction require inferring *missing* information about graph data, and in many ways, those two tasks are the graph analogues of supervised learning. *Community detection*, on the other hand, is the graph analogue of unsupervised clustering.

Suppose we have access to all the citation information in Google Scholar, and we make a *collaboration graph* that connects two researchers if they have co-authored a paper together. If we were to examine this network, would we expect to find a dense "hairball" where everyone is equally likely to collaborate with everyone else? It is more likely that the graph would segregate into different *clusters* of nodes, grouped together by research area, institution, or other demographic factors. In other words, we would expect this network—like many real-world networks—to exhibit a *community* structure, where nodes are much more likely to form edges with nodes that belong to the same community.

This is the general intuition underlying the task of community detection. The challenge of community detection is to infer latent community structures given only the input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The many real-world applications of community detection include uncovering functional modules in genetic interaction networks [Agrawal et al., 2018] and uncovering fraudulent groups of users in financial transaction networks [Pandit et al., 2007].

1.2.4 Graph classification, regression, and clustering

The final class of popular machine learning applications on graph data involve classification, regression, or clustering problems over entire graphs. For instance, given a graph representing the structure of a molecule, we might want to build a regression model that could predict that molecule's toxicity or solubility [Gilmer et al., 2017]. Or, we might want to build a classification model to detect whether a computer program is malicious by analyzing a graph-based representation of its syntax and data flow [Li et al., 2019]. In these *graph classification or regression* applications, we seek to learn over graph data, but instead of making predictions over the individual components of a single graph (i.e., the nodes or the edges), we are instead given a dataset of *multiple different graphs* and our goal is to make independent predictions specific to each graph. In the related task of *graph clustering*, the goal is to learn an unsupervised measure of similarity between pairs of graphs.

Of all the machine learning tasks on graphs, graph regression and classification are perhaps the most straightforward analogues of standard supervised learning. Each graph is an i.i.d. datapoint associated with a label, and the goal is to use a labeled set of training points to learn a mapping from datapoints (i.e., graphs) to labels. In a similar way graph clustering is the straightforward extension of unsupervised clustering for graph data. The challenge in these graph-level tasks, however, is how to define useful features that take into account the relational structure within each datapoint.