1 Predicting Missing Data in Networks

Most network data sets are *incomplete* in some way, because of how they were measured or because they are dynamic objects that change over time. But, if what we observe correlates with what is missing, then we may be able to predict the latter using the former.

Why are networks incomplete? Collecting complete network data can be expensive or impractical, and we may need to use what is available, rather than our ideal data. For every network, there are $\Theta(n^2)$ potential pairwise interactions among n nodes, and identifying with certainty which pairs are definitely connected, or definitely not connected, might require checking them all.

In biological networks, collecting data on interactions can require carrying out expensive laboratory experiments. For instance, to determine if two proteins bind to each other, we may need to test them *in vitro* and measure their binding strength experimentally. Or, deciding if two cells or two species interact may require careful observation *in vivo* or in the field. High-throughput laboratory methods are lowering the cost of some types of experiments, but $\Theta(n^2)$ remains an impractically large number when n is modest, even if individual experiments are relatively cheap.

Even in social networks, collecting network data can be expensive, e.g., offline social interactions can require observing individuals directly or asking them to name connections in a survey.[1] Digital systems mitigate these issues to some extent, but they induce other issues of missingness. For instance, a complete record of social interactions on one platform, like Instagram, is categorically missing all interactions that occurred on a different platform, or offline. If people use different platforms for different reasons or in different ways, then the way edges are missing will be platform-specific.

In networks, there are four main types of missing data prediction tasks, listed here in roughly ascending order of difficulty. We use the superscript $^\circ$ to denote an "observed" piece of data, e.g., E° is a set of observed edges, so that the un-superscripted variable denotes the actual data, e.g., E is the actual edge set, which includes any missing from E°.

1. **Predicting missing node attributes**
   Given a network $G = (V, E)$ and a partial annotation of nodes $\vec{x}^\circ$ by some attribute (categorical, scalar, or even vector), guess the missing attribute values.

2. **Predicting missing links**
   Given list of nodes and a partial list of edges $G^\circ = (V, E^\circ)$, guess which unconnected pairs i, j

[1] Such survey questions are called "name generators," because we ask a person to generate the names of their connections with certain properties. Unsurprisingly, humans are really bad at this, which is why backwards contact tracing is inherently hard — people fail to remember or misremember who their friends are, and who they name is shaped by recency, status, gender, aspirational, and subjective interpretation biases. For instance, see Marin, *Social Networks* **26**(4), 289-307 (2004).

among the set $X = V \times V - E^\circ$ are in fact missing connections, i.e., in the set $Y = E - E^\circ$.

3. **Predicting missing link attributes**
   Given a network $G = (V, E)$ and a partial annotation of edges $\vec{w}^\circ$ by some attribute (categorical, sign, scalar, or even vector), guess the missing attribute values.

4. **Predicting missing nodes**
   Given a partially observed list of nodes and the edges among them $G^\circ = (V^\circ, E^\circ)$, guess a set of missing nodes $V - V^\circ$ *and* their corresponding edges $E - E^\circ$.

The first three of these can be thought of as different kinds of *interpolation* or *imputation*, in which we are filling in missing pieces of the network. The fourth, however, is more akin to *extrapolation*, because we have to guess how to extend or grow the network. (A related, but easier, task is identified *spurious* links or nodes, i.e., links that are observed but do not actually exist. Can you see why detecting spurious links is easier than predicting missing links?)

2 Predicting missing node attributes

Suppose $G = (V, E)$ is a fully observed network (no spurious or missing links) whose nodes are annotated with some metadata $\vec{x}$. For simplicity, assume that the ith node's metadata x_i is just a single value (we'll consider vector values later).

This value might be *categorical*, in which case we may call it a node "label." For example, in a biological network of proteins, it might denote a node's function[2] as in catalysis or transport. Or, in a social network, it might denote a node's social or physical attribute, like gender or vaccine status. The value could also be *scalar*. For example, if nodes are species, x_i could denote i's abundance or body mass, or if nodes are people, x_i might be the person's age or reflect some preference.[3]

Because collecting data is expensive or hard, the set of metadata we observe $\vec{x}^\circ$ is incomplete, and some nodes are labeled with a missing value $x_i^\circ = \emptyset$. In the task of node attribute prediction, we ask the question

For each observed missing value $x_i^\circ = \emptyset$, how can we make a reasonable guess x_i^* of its actual value x_i, using only the network G and observed values $\vec{x}^\circ$?

[2] In molecular networks, such labels are often derived from the Gene Ontology, which provides rich, but incomplete annotations of various biological molecules: `http://geneontology.org/docs/ontology-documentation/`.

[3] If the metadata are vector-valued, then what is missing could be an arbitrary subset of elements of each vector, or the same single element of every vector, or entire vectors for a subset of nodes. The problem is still one of imputation, and one can use sophisticated Bayesian techniques to model the correlation between a latent variable representation of the network and the attributes in order to guess the missing vector elements. For instance, see Fosdick and Hoff, *J. Am. Stat. Assoc.* **110** (2015).

From an optimization perspective, we are aiming to minimize the difference between the predicted values x_i^* and the actual missing values x_i.[4]

2.1 The missingness function f

The observed metadata $\vec{x}^\circ$ and the actual metadata $\vec{x}$ are related to each other via a *missingness function* f, which chooses the particular subset we observe $\vec{x}^\circ = f(\vec{x})$.

Knowing something about the structure of f, i.e., how it tends to choose which node values it reveals and which it hides (e.g., its biases), or knowing something about how $\vec{x}$ is distributed across the network, often allows us to build better attribute prediction algorithms. In the most naïve case, we assume that f is unknown and unknowable, which leads us to a baseline algorithm for predicting missing node attributes.

A baseline algorithm. In the absence of any information about f or about how the values of x_i° correlate with each other across the network, a simple **baseline prediction** algorithm is to say

$$\text{if } x_i^\circ = \emptyset, \qquad x_i^* = \text{Uniform}(\vec{x}^\circ - \emptyset) \ , \tag{1}$$

that is, we "impute" each missing value as a uniform draw from the empirical distribution of non-missing values. This baseline works for all types of metadata, categorical, scalar, and vector-valued; in the latter case, we would use the empirical distribution of non-missing values in the jth element to impute missing values in the jth dimension.

If a correlation exists between network G and its metadata $\vec{x}$, then we can likely improve over this algorithm, either by learning a model of the relationship between f and G, or by designing a predictor based on assumptions about f and G. There are, of course, many ways we could do this, and these largely fall into two categories:

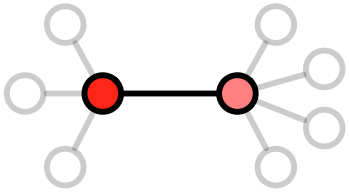
- *Local* predictors make a prediction about x_i based only on the metadata associated with nodes close to node i, e.g., nearest neighbors. Local predictors are often computationally light-weight, and are based on theoretical assumptions about f.
- *Global* predictors integrate information across the entire network G to make a prediction about x_i. Global predictors are often computationally expensive, are more agnostic about f, and are based on more sophisticated models, including graph regularization, probabilistic models, graph embeddings, or neural networks.

We'll focus our coverage on local predictors, to build intuition, and close with some discussion of global predictor methods.

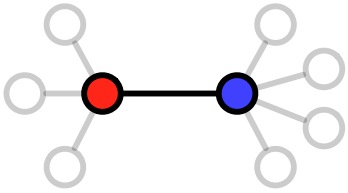
[4] How we quantify that difference will depend both on the type of attributes (categorical, scalar, vector, etc.) and how much weight we assign to different kinds of errors; in the agnostic case, we might assume equal weight for every error, but such an assumption is an implicit valuation and comes with specific ethical implications.

2.2 Assortative mixing and local smoothing

A common feature of many networks is that node annotations are **assortative**,[5] meaning that given an edge (i, j), the attributes of those nodes x_i and x_j will tend to be more similar to each other than will the attributes of an unconnected pair i, j. In other words, attributes correlate across edges, or "like links with like." The opposite would be **disassortative** mixing, in which attributes on either end of an edge are more dissimilar than those of unconnected pairs, or, "like links with dislike." This insight allows us to construct a *local smoothing* algorithm for predicting missing attributes.



assortative mixing



disassortative mixing

Specifically, we can

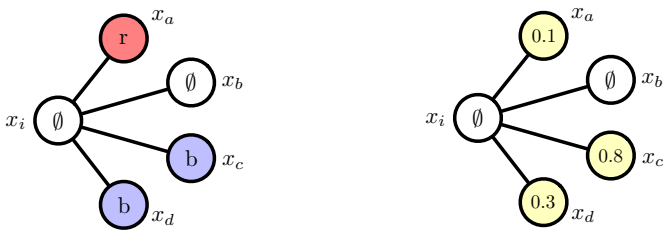
predict a missing attribute to be the average (scalar) or
most common (categorical) value of its neighbors' non-missing attributes.

To make this work in practice, we first define a "neighborhood" function $\nu(i)$, which returns the set of i's neighboring nodes. This function will let us select out of the global $\vec{x}^\circ$ the labels that are local to i, denoted $\{x^\circ_{\nu(i)}\}$. Given these local labels, we could apply any function g to assign a label to x_i, so long as g can take a variable number of labels as input (do you see why this is necessary?). Setting g to be a function like the `mean` (for scalar values) or the `mode` (for categorical values) will leverage the assortative mixing of attributes to make better than baseline predictions. Note, however, that g cannot be applied to a set that includes missing attributes itself, i.e., if any of i's neighbors are also missing a label. The solution is simply to exclude these missing values, i.e., to only apply g to the set $\{x^\circ_{\nu(i)}\} - \emptyset$. Hence, we say

$$\text{if } x_i^\circ = \emptyset, \qquad x_i^* = \begin{cases} \text{mean}\Big(\{x^\circ_{\nu(i)}\} - \emptyset\Big) & \text{if } x \text{ is scalar} \\ \text{mode}\Big(\{x^\circ_{\nu(i)}\} - \emptyset\Big) & \text{if } x \text{ is categorical} \end{cases} .$$

If *all* of i's neighbors have missing values, and thus $\{x^\circ_{\nu(i)}\} = \emptyset$, we can revert to the baseline prediction algorithm of Eq. (1). And finally, predictions for multiple nodes with missing values should be made in parallel, rather than in sequence. (Do you see why?)

[5] This pattern can also go other names, and is commonly called *homophily* in the context of social networks.



Consider the two little examples below. Here, $\nu(i)$ returns $\{a, b, c, d\}$, and hence the set of neighboring labels of i is $\{x^{\circ}_{\nu(i)}\} = \{\mathrm{r}, \emptyset, \mathrm{b}, \mathrm{b}\}$ on the left, and $\{x^{\circ}_{\nu(i)}\} = \{0.1, \emptyset, 0.8, 0.3\}$ on the right. The local smoothing algorithm would then predict $x_i^* = \mathrm{b}$ and $x_i^* = 0.4$, respectively.

2.3 Measuring performance: the confusion matrix

A crucial step in assessing the usefulness of any prediction algorithm is measuring its performance. How we do this differs depending on whether the metadata values are categorical or scalar, and the weight we assign to different kinds of errors. Because they require a bit more explanation, we'll focus on categorical variables, and then touch on scalar values at the end of this section.

Categorical values. In this setting, the fundamental tool we use to measure performance is called a **confusion matrix** $\mathcal{C}$. Without loss of generality, let the attributes be $x_i \in \{1, 2, \ldots, c\}$, i.e., there are c distinct values a metadata variable can take. (Imagine simply relabeling the actual values as $1 \ldots c$ for convenience.) Each input we apply our algorithm $\mathcal{A}$ to then has a *predicted label* p and an *actual label* q. If $p = q$, then the prediction is correct. But, if $p \neq q$, then the algorithm has made a mistake.[6]

A confusion matrix tabulates the $c \times c$ pairwise results of these predicted and actual labels:

$$\mathcal{C}_{pq} = \text{the number of inputs with (\textit{predicted} label } p\text{) and (\textit{actual} label } q\text{)} \ . \tag{2}$$

The diagonal $\mathcal{C}_{pp}$ counts the correct predictions, the off-diagonal elements count the mistakes, and the column sums give the actual frequencies of each label in the missing data.

A confusion matrix $\mathcal{C}$ provides complete information on the behavior of a prediction algorithm: it tells us how often $\mathcal{A}$ confuses a q (actual value) for a p (predicted value), and it also tells us how many q's there actually are.

Often, we want to summarize the contents of $\mathcal{C}$ in a single number that describes the algorithm's "overall" performance, because c^2 numbers can be a lot. There are *many* different ways to sum-

[6] When $c = 2$ (binary classification), we can use the language of true and false, positives and negatives.

marize a $\mathcal{C}$ matrix, and each emphasizes slightly different aspects of $\mathcal{A}$'s behavior.[7] A common measure is the **accuracy**, defined as

$$\text{accuracy (ACC)} = \frac{\text{number of correct predictions}}{\text{number of inputs}} = \frac{\sum_p \mathcal{C}_{pp}}{\sum_{p,q} \mathcal{C}_{pq}} = \frac{1}{N} \sum_p \mathcal{C}_{pp} \ , \tag{3}$$

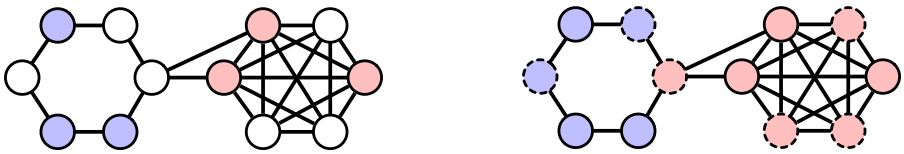
where N is the number of inputs on which a prediction was made.

Warning: accuracy (ACC) is most useful when label frequencies are relatively balanced. If some labels are far more common than others, accuracy can be high even if the algorithm's performance is poor. For instance, imagine we have $c = 2$ labels (a "binary" classification task), and label 1 occurs 90% of the time. An algorithm that always guesses "1" will have an ACC = 0.90, but is correct 0% of the time on the minority label. That might seem like a pretty good accuracy, but it's a terrible prediction algorithm.

Scalar values. When metadata values are scalar, we can simply (i) make a scatter plot of the predicted x_i vs. the actual x_i (which is comparable to the confusion matrix), and (ii) report the r^2 correlation (or similar summary statistic) between the two (which has similar weaknesses as ACC).

2.4 A simple example

Consider the partially observed network below, on the left, which has $n = 13$ nodes and 6 nodes are labeled already. Suppose that the actual labeling $\vec{x}$ is such that the left half of the network is all blue, and the right half is all red.



Applying the local smoothing algorithm produces the network on the right (do you see why, for each node?), which makes one mistake. The resulting confusion matrix is then

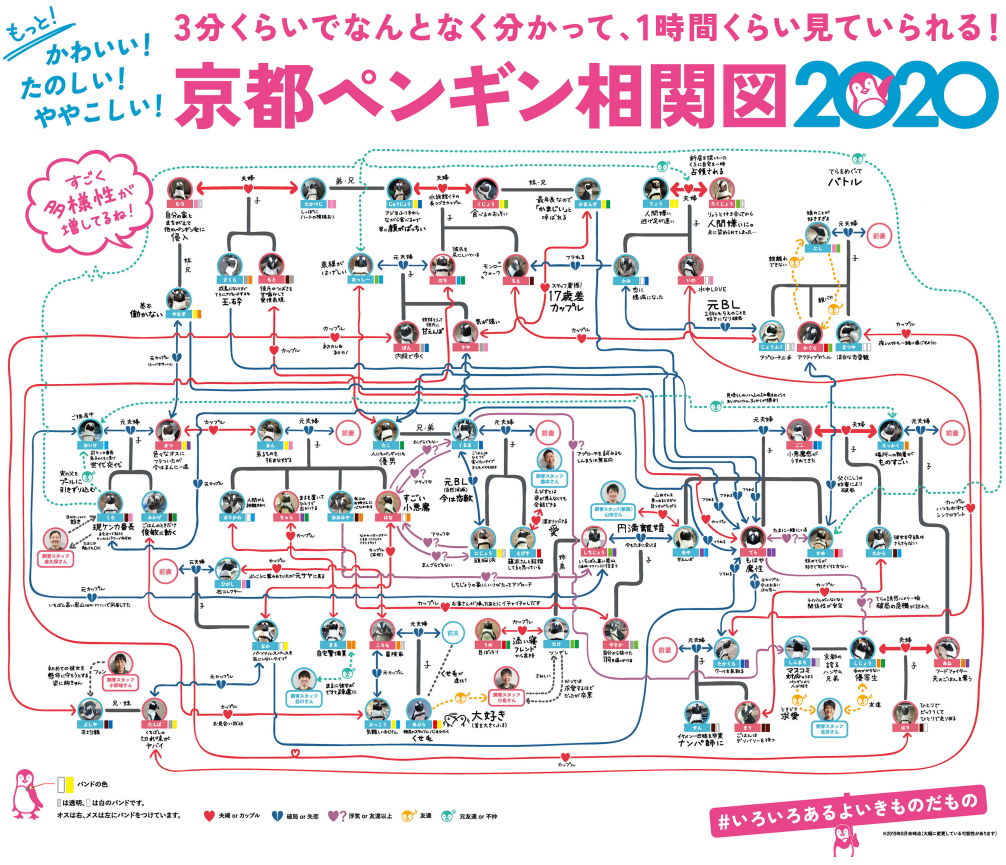
		actual	
	$\mathcal{C}$	r	b
pred.	r	3	1
	b	0	2

corresponding to an accuracy of ACC = 5/6 = 0.83.

[7] The F_1 *statistic* is common in machine learning, but, like accuracy, is also sensitive to class imbalance. The *Informedness* has better properties in such cases. See `https://en.wikipedia.org/wiki/Confusion_matrix`

2.5 The Kyoto Aquarium's Penguins

As a more realistic, but also more complicated example, consider the network of relationships among penguins at the Kyoto Aquarium, as recorded in 2020, and transcribed into a network by Heather Brooks and Michelle Feng, with help from Hiroki Sayama.[8] In this multiplex animal social, nodes are penguins, and edges represent different kind of social relationships (couples, exes, "it's complicated," friends, enemies, and family). Taking the union of the different edge sets produces a network with $n = 59$ nodes and $m = 91$ directed edges.



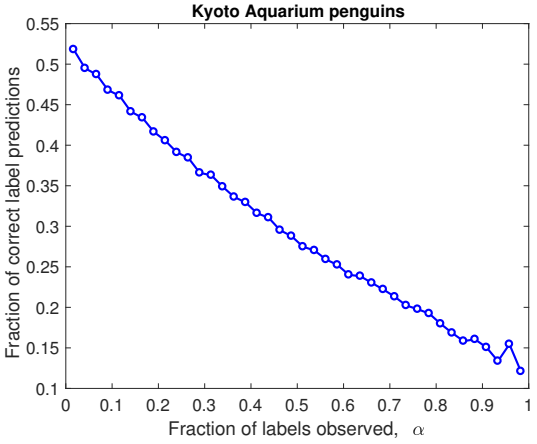
If you squint at the Kyoto Aquarium visualization, you can see that most relationships are undirected or "bidirectional," but not all of them. Node attribute prediction works just as well across directed edges as undirected edges—the only difference is that the neighborhood set $\nu(i)$ for some i

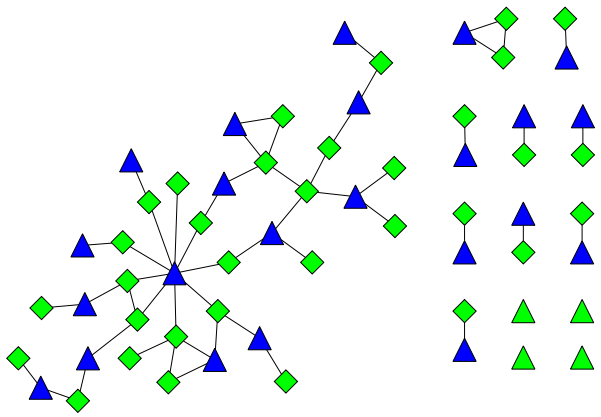
[8] See Brooks and Feng, "Penguins of Kyoto Multilayer Network." `https://bitbucket.org/mhfeng/penguins_of_kyoto/src` (2020).

may not equal the neighborhood set $\nu(j)$ for some j, because the edge points in one direction, but not the other. The neighborhood function remains perfectly well-defined, and so local smoothing can be applied.

The node metadata for this network is mainly the biological sex of the penguin, which is recorded as a binary variable, with 36 males and 23 females. To apply the local smoothing predictor, we set up a simple numerical experiment where we (i) choose a uniformly random fraction $\alpha \in (0, 1)$ of labels to observe, (ii) apply the local smoothing predictor to a number of networks at each choice of α and, (iii) record the average accuracy (fraction of correct label predictions). In this kind of experiment, we usually expect the accuracy to be close to a baseline when $\alpha \approx 0$, meaning we observe almost no labels and most nodes have neighborhood sets that contain only missing values.

The results bear this out below, but then do something surprising: the accuracy gets *worse* the more labels we observe (larger α). How can this be? The answer is that in this network, social relationships are largely *disassortative*, with most connections being between male and female penguins (but not always), and so local smoothing does exactly the opposite of what we want. We can see the disassortative mixing pattern clearly by visualizing the network, or, we could calculate the assortativity coefficient for the metadata, which would yield a negative value.





3 Advanced approaches

Warning: this material is rough and still under development.

Using the network to fill in the missing node attributes only works if there is some correlation between the missing attribute of a node i and the non-missing attributes of other nodes in the

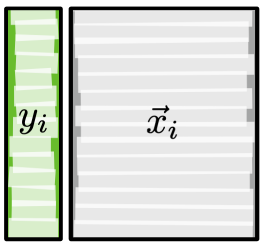
network. If we can automatically identify which nodes those are, then we have an algorithm by which to make a more accurate guess of node i's missing attribute.

Although node attribute predictions can work on correlations alone—we have no idea or don't care why node attributes covary with edges—we will usually get better, more accurate predictions if we think a little about the causal structures that make node attributes covary with edges. Basically, if the network is independent of, or rather not causally related to, node attributes, then using the network as part of our prediction algorithms won't improve our predictions, and vice versa.

3.1 A modern approach: recast as machine learning

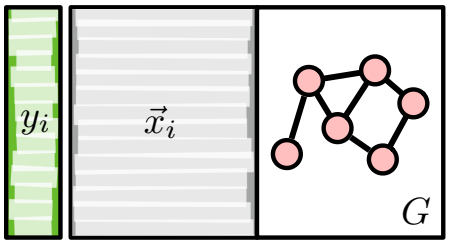
From this perspective, we can recast the problem of predicting missing node attributes as a kind of standard machine learning prediction problem, which allows us to use many standard ML tools to predict missing information.

First, suppose we ignore the network entirely: we have n nodes and each node i is associated with both a set of covariates $\mathbf{x}_i$ (bold indicates a vector) and some outcome variable of interest y_i, some of which are observed and some are missing. Ignoring the network G, we can use standard ML tools, like a random forest or even simple multiple linear regression, to learn a model $y_i = g(\mathbf{x}_i)$, which takes as input a node's attributes $\mathbf{x}_i$ and predicts its outcome y_i (figure below, left panel). This works because we're ignoring the network, which tells us how the nodes are related to each other, and instead exploit the fact that the data $\{y_i, \mathbf{x}_i\}$ are just a standard set of points in a d-dimensional space (where d is the length $\mathbf{x}_i$). The model would be trained on the subset of nodes for which we have both an outcome variable and the corresponding attributes, and then we would apply that model to the unlabeled nodes to predict their missing values y_i using their observed covariates $\mathbf{x}_i$.



predict outcomes from attributes

$$y_i = g(\vec{x}_i)$$



predict outcomes from attributes and graph

$$y_i = g(\vec{x}_i, G)$$

But, if nodes are assortative (or disassortative) in their attributes within the network, then the pairwise information in the network would tell us which rows in the matrix are more, or less, correlated with each other, and ignoring those correlations makes our predictions strictly worse than if

we incorporated them, somehow. This is the effect that the local smoothing predictor exploited to produce more accurate predictions of missing attributes y_i than baseline guessing. In other words, local smoothing predictors are a kind of unsupervised model, where we ignore node-level covariates $\mathbf{x}_i$ and only use G to predict y_i. The more general approach aims to learn a model $y_i = g(\mathbf{x}_i, G)$ that takes as input both a node's attributes $\mathbf{x}_i$ and the network G, and predicts its outcome y_i (figure above, right panel).

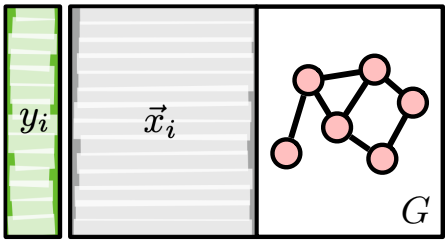
Hence, for predicting missing node attributes there are two sources of information about y_i:

1. the node-level covariates $\mathbf{x}_i$, which may or may not correlate with y_i, and
2. the location of i in the network G, relative to other nodes and their outcomes y_j, which may or may not correlate with y_i.

In real data, it's possible that both $\mathbf{x}_i$ and G correlate with y, or that only one or the other does. Usually, both correlate because both are generated by the same underlying data generating process.

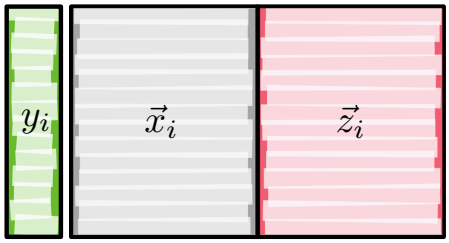
The purest way to model a network where nodes have attributes $(y_i, \mathbf{x}_i)$ is a probabilistic generative model that jointly generates both the node attributes and the network edges, in a way that allows us to vary the correlational structure among each.[9]

An alternative and very a popular approach is to instead transform the network G into another set of node covariates z_i which recasts the problem back into the standard machine learning paradigm of points in a high-dimensional space. That is, encode the network's structure *from i's perspective* into a set of new "graph features" for i, i.e., we make z_i a set of node-level summary statistics that encode information about the "position" of i in G.



predict outcomes from attributes and graph

$$y_i = g(\vec{x}_i, G)$$



predict outcomes from attributes and graph features

$$y_i = g(\vec{x}_i, \vec{z}_i)$$

There are two main ways we can operationalize this approach. The first uses the same node-level summary statistics we've seen before, e.g., a node's degree, node-level clustering coefficients, mea-

[9] For instance, see Newman and Clauset, *Nat. Commun.* **7**, 11863 (2016) and Hric et al. *Phys. Rev. X* **6**, 031038 (2016).

sures of position like eccentricity, and a whole set of measures called "centrality" scores, which are meant to capture various notions of a node's position (and relative importance) in the structure of the network.[10] The upside of this approach is that such network measures are "native" to the network, being defined explicitly in terms of the graph's structure, and are hence relatively interpretable. The downside is that many of these measures are computationally expensive, and calculating them for large networks may be impractical or even impossible. In addition, these measures may not capture all the different or important aspects of the structure of G.

The second transforms the graph G into a set of node-level "coordinates" z_i in some high-dimensional space $\mathbb{R}^d$, thereby "embedding" the nodes in that space, such that nodes that are close to each other in $\mathbb{R}^d$ are "similar" in some way. If we do this in a way that captures the way the outcome variable y_i covaries with the network's structure, then these coordinates z_i will be useful in predicting the outcome variables y_i, in conjunction with the node attributes $\mathbf{x}_i$.

The most popular way to do this is to use a "node embedding" approach, and within that, the most popular way to produce the embedding is with a "graph neural network," which uses a neural network to transform a corpus of short walks (sequences of nodes) on the network into a set of n spatial coordinates, one for each node, representing an "embedding" of the n nodes into some high dimensional space $\mathbb{R}^d$.

Non-neural network approaches to embedding use different methods to convert the graph to coordinates, e.g., GLOVE, that itself uses word embedding techniques from natural language processing (NLP) to do the embedding part itself. Since these techniques are designed for language, we can use them as a subroutine if we can develop a method for transforming the graph G into a corpus of "sentences." We'll cover two simple techniques for doing this.

3.1.1 Converting a network structure into node attributes

[[This section still in development]]
GLOVE
DeepWalk
word2vec

[10] Centrality measures are very popular in the sociology branch of network science, in part because they are often grounded in some kind of social theory about power or importance, as mediated by location in a network. These social theories are typically dynamical in nature, meaning they are based on some notion of cause and effect. But centrality measures are usually structural, meaning we calculate them as $z_i = g(G, i)$. this difference is basically a computational compromise: it's expensive to simulate the full dynamical process in order to calculate z_i, and so instead, we just calculate z_i from the static network G. (Notably, there are some centrality measures that are dynamical, and a few select others where one can show that the structural calculation analytically produces the same result as the dynamical simulation). Popular examples of centrality measures include degree centrality, harmonic centrality, and betweenness centrality.

Residual2Vec