1 Predicting Missing Data in Networks

Most network data sets are *incomplete* in some way, either because of the way we measured them or because they are dynamic objects that change over time. But, if what we do observe correlates with what is missing, or what may occur in the future, then we may be able to predict one using the other.

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 subscript $\circ$ to denote an "observed" piece of data, e.g., E° is a set of observed edges, so that the un-subscribed 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 $Y = V \times V - E^\circ$ are in fact missing connections, i.e., in the set $X = E - E^\circ$.

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 of these three 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 race. The value could also be *scalar*, e.g., if nodes are species, x_i might give that species' abundance or body mass, or if nodes are people, x_i might be the person's age or reflect some preference.

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 $\vec{x}^\circ$?

From an optimization perspective, we are aiming to minimize the difference between the predicted values x_i^* and the actual missing values x_i.[3]

[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] 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.

2.1 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

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

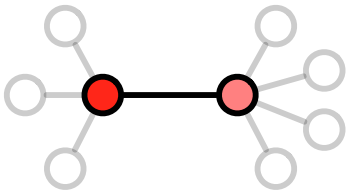
that is, we "impute" each missing value as a uniform draw from the empirical distribution of non-missing values.

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, which make a prediction about x_i based only on the local neighborhood of the node i, e.g., its nearest neighbors and their values, and *global* predictors, which integrate information across the entire network G to make a prediction about x_i.

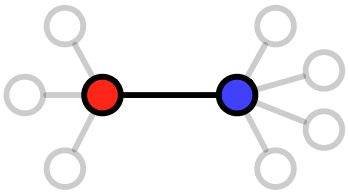
2.2 Assortative mixing and local smoothing

A common feature of many networks is that node annotations are **assortative**,[4] 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.

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



assortative mixing



disassortative mixing

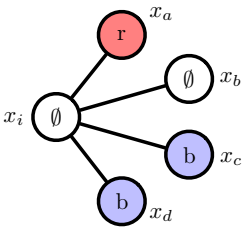
Specifically, we can

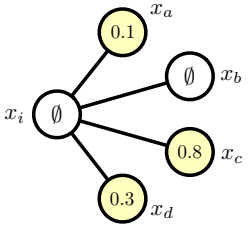
predict a missing attribute to be the average (scalar) or most common (categorical) value of its neighbors' non-missing attributes.

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

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

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?)





Consider the two little examples above. Here, $\nu(i)$ returns $\{a, b, c, d\}$, and hence the set of neighboring labels of i is $\{x^{\circ}_{\nu(i)}\} = \{\text{r}, \emptyset, \text{b}, \text{b}\}$ 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 $x^{*}_i = \text{b}$ 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.[5]

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 summarize a $\mathcal{C}$ matrix, and each emphasizes slightly different aspects of $\mathcal{A}$'s behavior.[6] 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

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

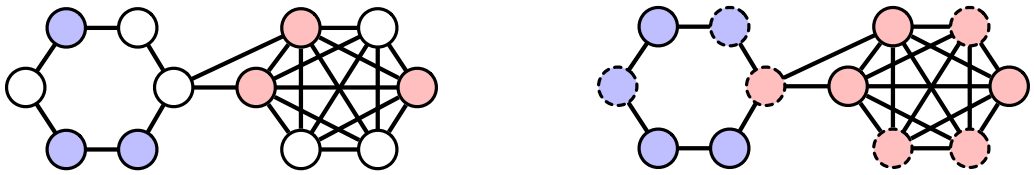
[6] The F_1 *statistic* is common in machine learning, but, like accuracy, is also sensitive to class imbalance. The *Informedness* has better properties in such cases. See `https://en.wikipedia.org/wiki/Confusion_matrix`

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 An 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.

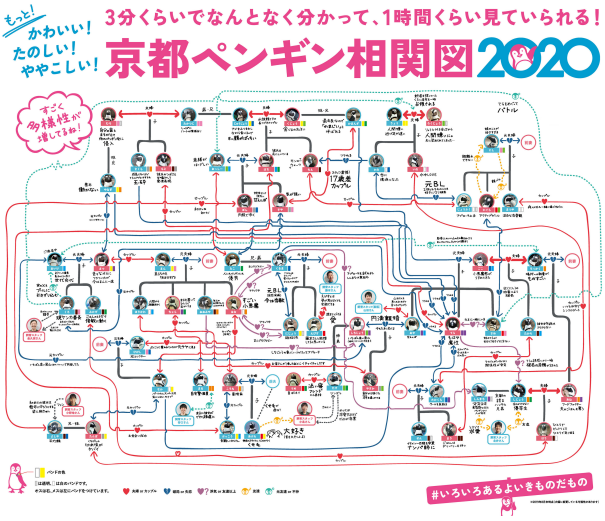


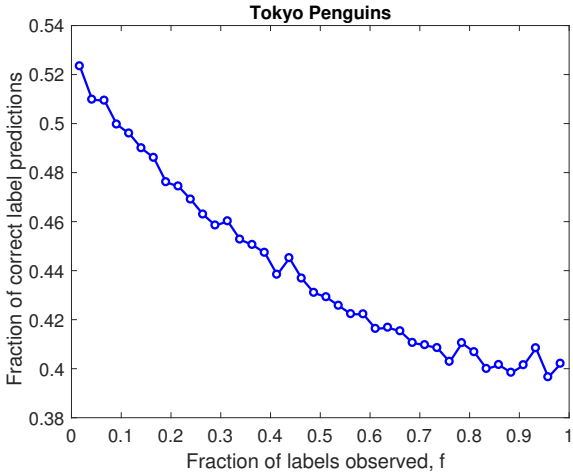
Applying the local smoothing algorithm produces the network on the right (do you see why, for each node?), which makes one mistake. The resulting confusion matrix is then

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

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

2.5 An example: Tokyo Zoo's Penguins


もっと!
かわいい!
たのしい!
ややこしい!
3分くらいでなんとなく分かって、1時間くらい見ていられる!
京都ペンギン相関図2020
すごく多様性が増してるね!
#いろいろあるよいきものだもの


Tokyo Penguins
Fraction of correct label predictions
Fraction of labels observed, f
0.54
0.52
0.5
0.48
0.46
0.44
0.42
0.4
0.38
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1