1 Predicting Missing Data in Networks

Most network data sets are *incomplete* in some way. However, 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. 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.

The two most common missing-data prediction tasks are *node attribute prediction* and *missing link prediction*. Let the superscript $^\circ$ to denote an "observed" piece of data, e.g., $\vec{x}^\circ$ is a set of observed node attributes and 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 among the set $X = V \times V - E^\circ$ are in fact missing connections, i.e., in the set $Y = E - E^\circ$.

Both of these can be thought of as a form of *interpolation* or *imputation*, in which we are filling in missing pieces of the network. Here, we will learn about the general parameters of these two prediction tasks, describe a few simple approaches for solving them, and learn how to assess their accuracy in practice.

2 Predicting missing node attributes

Suppose $G = (V, E)$ is a fully observed network (no spurious or missing links) whose nodes are annotated with metadata $\vec{x}$, where the ith node's metadata x_i is just a single value.[1]

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.

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, for each observed missing value $x_i^{\circ} = \emptyset$, we seek to make a guess x_i^* using only the network G and observed values $\vec{x}^{\circ}$. And ideally, these guesses minimize the difference between the predicted values x_i^* and the actual missing values x_i.[3]

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.

[1] Vector-valued metadata can exhibit more complicated forms of missingness, and hence typically require sophisticated techniques.

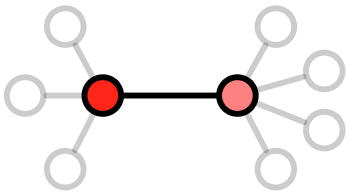
[2] In molecular networks, such labels are often derived from the Gene Ontology, which provides rich, but incomplete annotations of various biological molecules: `http://geneontology.org/docs/ontology-documentation/`.

[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.

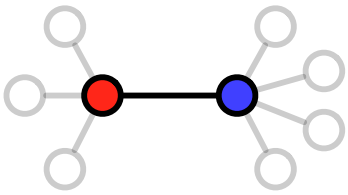
If a correlation exists between network G and its metadata $\vec{x}$, then we can likely improve over this algorithm, either by learning a model of the relationship between f and G, or by designing a predictor based on assumptions about f and G. There are, of course, many ways we could do this, and these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



assortative mixing
(links between similar)



disassortative mixing
(links between dissimilar)

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

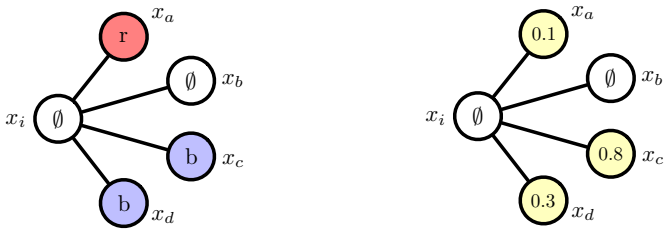
[4] This pattern can also go other names, and is commonly called *homophily* in the context of social networks.

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?)

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

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

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

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 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

[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`

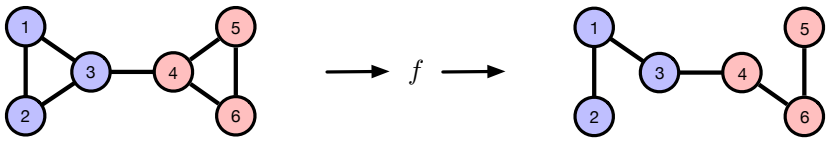
		actual	
	$\mathcal{C}$	r	b
pred.	r	3	1
	b	0	2

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

3 Predicting missing links

Imagine an unobserved simple network $G = (V, E)$. However, because collecting interaction data is expensive, only a subset $E^\circ \subset E$ of actual connections is observed, and a *missingness function* f chooses which subset that is. If there are some patterns within the observed edges E° that correlate with with the actual edges E, then we may be able to predict (better than baseline guessing) which of the observed unconnected pairs $Y = V \times V - E^\circ$ are in fact missing links $X = E - E^\circ$.

Here is a small example of this process of a missingness function f being applied to an actual graph G (on the left) to produce the observed network G° (on the right), where the removed lines are the missing links.



actual $G = (V, E)$ observed $G^\circ = (V, E^\circ)$

Predicting missing links (the removed lines in the above example) is a much harder problem than predicting missing node attributes. With node attributes, we start off knowing which attributes are missing $x_i = \emptyset$, but with missing links, we do not. Instead, our task is to sort among all the 0s of the adjacency matrix, which are the "non-edges" or unconnected pairs in the observed graph G°, to find those that should be 1s. Because most real-world networks are sparse, this task is like searching for $O(n)$ needles in a $\Theta(n^2)$ haystack. Hence, the baseline accuracy for guessing will be $O(1/n)$, making it extremely unlikely to guess correctly just by chance.

Link prediction methods operate by defining a *score* function over unconnected pairs $i, j \in X$, and the better the predictor, the more likely it will assign a higher score to a pair i, j that is actually a missing link.

A baseline algorithm. In the absence of any information about f or about any patterns within E°, a simple **baseline prediction** algorithm assigns a value that is independent of the graph G, meaning that every input pair i, j is equally likely to receive a particular score:

$$\text{if } i, j \in X \quad \text{score}(i, j) = \text{Uniform}(0, 1) \ , \tag{4}$$

that is, we assign a uniformly random score between 0 and 1 to each unconnected pair. If we apply this algorithm to the above example, and then *sort* the candidate pairs i, j by their score(i, j) values, we obtain a score table:

i	j	score(i, j)
1	5	r
1	4	r
1	6	r
2	3	r
2	6	r
2	4	r
2	5	r
3	5	r
3	6	r
4	5	r

where r is a stand-in for Uniform$(0, 1)$, and the ordering is arbitrary among tied scores. (Note that in practice, ties must be broken randomly. Do you see why?) You can see the two missing edges $Y = \{(2, 3), (4, 5)\}$ in the list, but there's nothing about their scores to suggest that they are any more or less likely to be the missing links than another pair.

If some correlation exists between the observed edges E° and the missing edges Y, then we can likely improve considerably over this algorithm. Just like with predicting missing node attributes, there are many ways we could do this, and these largely fall into three categories. We'll focus on one below, and then briefly describe the remaining two in Section 3.4.

3.1 Topological predictors and local smoothing

Topological predictors use the structure of the graph itself, from the perspective of nodes i, j as a way to create a score function that may correlate with the missing links. Each such predictor is based on a specific assumption about how edges form. For instance, we might score a pair by the number of common neighbors (the more, the better), the number of shortest paths between the nodes, the product of their degrees, etc.

Just as node attributes are often assortative (or, homophilous) in networks, edges themselves are often **assortative**, meaning that they tend to cluster together. This pattern is the idea behind the clustering coefficient, which measures the "local" density of edges. And, just as it did for node attributes, this pattern allows us to construct a kind of local smoothing algorithm for predicting missing links. Specifically, we can

predict a missing link to occur where it would cluster with other edges.

There are many ways to turn this idea into an algorithm for predicting missing links. We will explore two. One uses the **Jaccard coefficient**, which quantifies the degree of overlap among two nodes' neighborhoods. Using the same notation as in the Lecture on predicting missing node attributes, let $\nu(i)$ return the set of neighbors of node i. The Jaccard coefficient is defined as

$$\text{Jaccard}(i,j) = \frac{|\nu(i) \cap \nu(j)|}{|\nu(i) \cup \nu(j)|} \ , \tag{5}$$

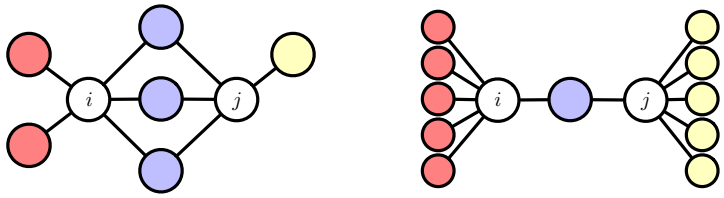
which measures the fraction of neighbors of either node that are neighbors of both nodes, i.e., the density of common neighbors. Two nodes with many common neighbors will have a higher Jaccard coefficient. In that case, adding an edge between i, j would then "close" each of the open triads i, k, j, which would increase both their local, and the global, clustering coefficient.

To turn this pairwise network measure into a prediction algorithm, we say

$$\text{score}(i,j) = \text{Jaccard}(i,j) + \text{Uniform}(0,\epsilon) \ , \tag{6}$$

where $\text{Uniform}(0,\epsilon)$ adds a small amount of noise in order to break ties randomly without changing the relative ordering outside those ties. (Do you see why this is necessary?)

Consider the following small examples. On the left, $\nu(i)$ returns 5 nodes, $\nu(j)$ returns 4 nodes,



and there are 3 common neighbors. Hence, the $\text{Jaccard}(i,j) = 0.50$, reflecting the relatively high proportion of total neighbors that i, j have in common. On the right, i, j have many fewer common neighbors, and their Jaccard coefficient is correspondingly smaller, only 0.091.

A second local topological predictor is the **degree product**, which embodies the idea we learned from our analysis of random graphs that nodes with high degrees are likely themselves to be connected, just by chance. We define this predictor as $\text{score}(i,j) = k_i k_j + \text{Uniform}(0,\epsilon)$. How would

degree-product scores differ from those of the Jaccard coefficient for the above examples?

Returning to our running example, with its two removed edges, and then applying the Jaccard and degree-product predictors to the observed network G', we obtain the following score tables:

i	j	Jaccard score(i,j)
4	**5**	$1/2 + r$
2	**3**	$1/2 + r$
3	6	$1/3 + r$
1	4	$1/3 + r$
1	5	r
1	6	r
2	6	r
2	4	r
2	5	r
3	5	r

i	j	degree product score(i,j)
1	4	$4 + r$
1	6	$4 + r$
3	6	$4 + r$
1	5	$2 + r$
2	**3**	$2 + r$
2	6	$2 + r$
2	4	$2 + r$
3	5	$2 + r$
4	**5**	$2 + r$
2	5	$1 + r$

where the missing links $Y = \{(2,3),(4,5)\}$ are highlighted, and links with the same score are ordered arbitrarily.

On the left, the Jaccard link predictor does very well at assigning these missing links high scores compared to the non-missing links, while on the right, the degree product seems very poor. These different behaviors illustrate a key point about link predictors: each predictor captures different types of correlations between observed and missing edges, and hence each will perform well on some inputs and poorly on others. In this case, the actual network has substantial local edge density, and so the Jaccard function does well. But the degree product predictor would perform better if the network had much higher variance in its degree structure.

To quantify these differences in performance, compare them to the baseline, or understand how close they might be to an *optimal* predictor, we need a compact way of quantifying their performance.

3.2 Measuring performance: the AUC

Predicting missing links is a binary classification problem: every candidate $i, j \in X$ is either a missing link or not.

To summarize the performance of a link prediction algorithm, we can use the **AUC** statistic,[7] [8] a

[7] AUC is short for "Area Under the Curve," where the "curve" here is the Receiver Operating Characteristic (ROC) curve. ROC curves were invented during World War 2 as a method for assessing the performance of radar at detecting enemy aircraft. Neato.

[8] There are other ways to quantify accuracy in this setting, e.g., the F1-measure, each of which has a slightly different way of weighting the cost of different types of errors.

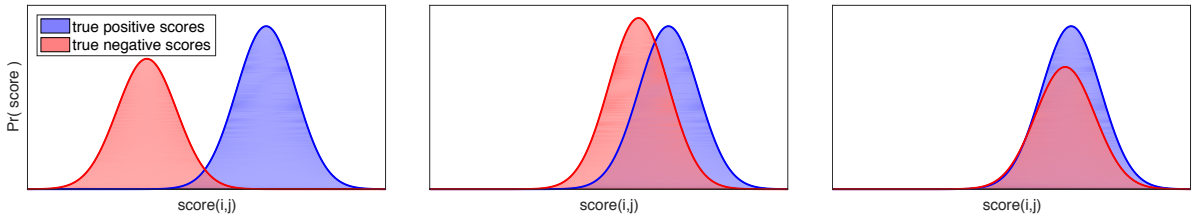
context-agnostic measure of its ability to distinguish a missing link $i, j \in Y$ (a true positive, or TP) from a non-edge $X - Y$ (a true negative, or TN). The AUC has two other attractive properties:

1. *scale invariance*, meaning it is not dependent on the scores themselves, and instead only depends on their relative values, and
2. *threshold invariance*, meaning it is a general measure of accuracy that doesn't require choosing a threshold on the scores for making predictions.

Mathematically, the AUC can be defined as

$$\text{AUC} = \Pr[\ \text{score(TP)} > \text{score(TN)}\ ] \quad . \tag{7}$$

That is, if we choose a uniformly random missing edge $a, b \in Y$ (TP) and a uniformly random non-edge $c, d \in X - Y$ (TN), the AUC is the probability that score(a, b) is higher than score(c, d).[9] The figure below illustrates this idea, for three hypothetical predictors. On the left, the predictor assigns scores in a way that the score distribution for true positives is well-separated from the score distribution of true negatives, which produces a high AUC value. In the middle example, the predictor assigns scores so that the distributions overlap more, which lowers the AUC. And, on the right, the predictor assigns scores so that the distributions are nearly the same, leading to a very low AUC.



AUC = 0.99 AUC = 0.76 AUC = 0.55

If the algorithm is no better than guessing at random, then AUC = 0.5 and it is as equally likely that score(TP) < score(TN) as it is that score(TP) > score(TN). If that's the case, the probability of the latter is 1/2, and hence that the AUC = 0.5. The baseline predictor of Eq. (4), which assigns a uniformly random value to every candidate, has exactly this behavior and thus has an AUC = 0.5. Hence, any algorithm with AUC > 0.5 does better than chance (the baseline). The maximum value of AUC = 1.0 is attained only when the algorithm assigns a higher score to every missing link (true positive) than it does to any non-edge (true negative).

[9] Fun fact: the AUC is mathematically equivalent to the Mann-Whitney U test and to the "probability of superiority," which is a concept about statistical effect sizes.

In practice, we compute a link prediction algorithm's AUC by numerically integrating its corresponding "ROC" curve to get, literally, the area under the curve.[10] Formally, the ROC curve is a parametric plot of the *true positive rate* (TPR) and the *false positive rate* (FPR), as a function of prediction threshold. The idea is straightforward. Imagine drawing a line across one of our score tables just below the ℓth row, and then sliding it up, or down the table. This line represents a prediction "threshold": all the rows ℓ and above we predict to be missing links, while all rows $\ell+1$ and below we predict to be non-missing links. The TPR is the fraction of all the actual missing links that are in the predicted-to-be-missing set (at or above row ℓ), and the FPR is the fraction of all the actual non-missing links (non-edges) that are in that same set. Plotting TPR(ℓ) vs. FPR(ℓ) sweeps out the ROC curve on the unit square, starting at $(0,0)$ and ending at $(1,1)$.

To calculate the AUC, we begin by augmenting the score table with three additional columns: a column τ that gives the *actual* status of the ℓth row (pair i,j) as either a true positive ($\tau_\ell = 1$) or true negative ($\tau_\ell = 0$), and then two columns, one for TPR(ℓ) and one for FPR(ℓ). If a predictor performs well, then in the τ column, the 1s will tend to be located higher up in the table, while if it is no better than baseline, the 1s will be scattered uniformly at random within this column.[11]

Once the τ column has been added, we can fill in the TPR and FPR columns simultaneously in a single scan down the table. Let $\mathcal{T} = |Y|$ be the total number of true positives (missing links; equal to the column sum of τ), and let $\mathcal{F} = |X - Y|$ be the total number of true negatives (non-edges; equal to the length of the table minus the sum of τ). The TPR(ℓ) and FPR(ℓ) are then defined as

$$\text{TPR}(\ell) = \frac{1}{\mathcal{T}} \sum_{k=1}^{\ell} \tau_k \qquad \text{FPR}(\ell) = \frac{1}{\mathcal{F}} \sum_{k=1}^{\ell} 1 - \tau_k \ . \tag{8}$$

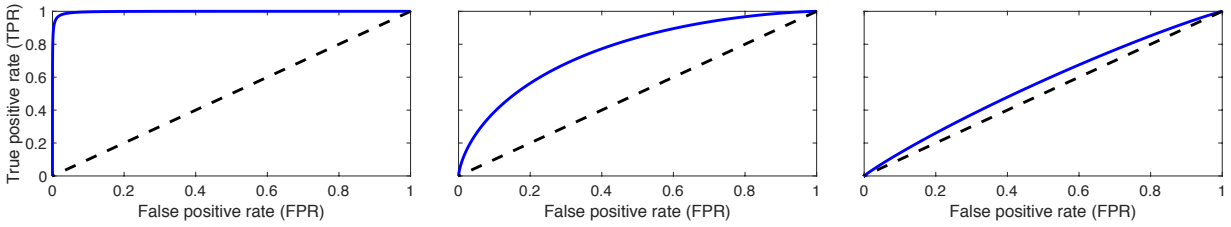
Using the same three sets of hypothetical score distributions we saw above, we can now compute the corresponding TPR and FPR functions, and then plot them against each other to obtain the ROC curve for each case. As a reference, each plot also shows the TPR = FPR line, which represents the baseline predictor's performance.

The area under each ROC curve is each hypothetical predictor's AUC, which we calculate by numerically integrating the ROC curve, using a simple box-rule approximation[12] technique, running

[10] We can also estimate the AUC via Monte Carlo, by evaluating Eq. (7) directly: sample many pairs of true positives and true negatives, and estimate the fraction of draws for which the former is assigned a higher score than the latter.

[11] If the 1s tend to be stacked at the bottom of the τ column, then the predictor is, in fact, pretty good at distinguishing 1s from 0s, it just gets it backwards. We can then construct a better-than-baseline predictor out of a worse-than-baseline predictor by simply doing the opposite of what it says (or by inverting its output scores).

[12] Recall the box rule for numerical integration: given a function $f(x)$, a range $[x_s, x_t]$, and an increment Δx, the integral $\int_{x_s}^{x_t} f(x)\mathrm{d}x \approx \sum_{x=x_s}^{x_t} f(x) \times \Delta x$. The trapezoid rule is slightly more accurate, although the difference is negligible when Δx is small, as is the case in link prediction (do you see why?).



AUC = 0.99 AUC = 0.76 AUC = 0.55

down the elements of the TPR and FPR vectors:

$$\text{AUC} = \sum_{\ell=1}^{|X|} \text{TPR}(\ell) \times [\text{FPR}(\ell) - \text{FPR}(\ell-1)] \quad , \tag{9}$$

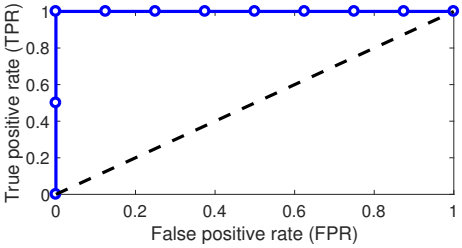
where we define $\text{FPR}(0) = 0$. Note that sometimes the area of a box we're adding up will be zero (do you see when this will happen?), and that's okay.

3.3 An example

To see how this works in practice, let's consider again the Jaccard and degree-product link predictors from Section 3.1. The two score tables below now include the three new columns: the actual status τ, the corresponding true positive rates (TPR), and the corresponding false positive rates (FPR) for a threshold drawn between that row and the next. (Exercise: verify that the columns were calculated correctly.)

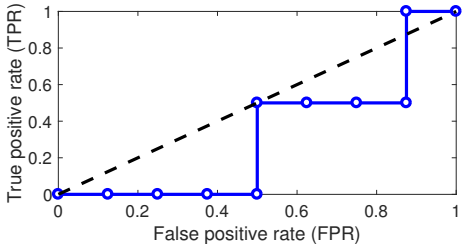
From these, we can plot the corresponding ROC curves, and apply Eq. (9) to calculate the AUC for each predictor. Unsurprisingly, the Jaccard predictor achieves perfect performance, with an AUC = 1.00. In this network, the degree-product predictor is in fact *worse* than guessing at random, producing an AUC = 0.31. This value is not as precise as it seems, however. Recall that the value r in the score column is a random variable. This means that the positions of the two true positives in the degree-product table will vary within the block of $\text{score}(i,j) = 2 + r$ rows, and these lead to slightly different TPR and FPR functions. The right thing to do here is to compute an *average AUC*, when these ties are broken randomly, which produces a more reliable estimate of the AUC. The highest value possible, if the ties break luckily for the missing links is 0.62, while the lowest, is 0.12 (do you see why?). The average is AUC = 0.37.

i	j	τ_k	TPR_ℓ	FPR_ℓ	Jaccard score(i,j)
4	5	1	0.5	0.0	$1/2 + r$
2	3	1	1.0	0.0	$1/2 + r$
3	6	0	1.0	0.125	$1/3 + r$
1	4	0	1.0	0.250	$1/3 + r$
1	5	0	1.0	0.375	r
1	6	0	1.0	0.500	r
2	6	0	1.0	0.625	r
2	4	0	1.0	0.750	r
2	5	0	1.0	0.875	r
3	5	0	1.0	1.00	r



Jaccard coefficient, AUC = 1.00

i	j	τ_k	TPR_ℓ	FPR_ℓ	degree product score(i,j)
1	4	0	0.0	0.125	$4 + r$
1	6	0	0.0	0.250	$4 + r$
3	6	0	0.0	0.375	$4 + r$
1	5	0	0.0	0.500	$2 + r$
2	3	1	0.5	0.500	$2 + r$
2	6	0	0.5	0.625	$2 + r$
2	4	0	0.5	0.750	$2 + r$
3	5	0	0.5	0.875	$2 + r$
4	5	1	1.0	0.875	$2 + r$
2	5	0	1.0	1.000	$1 + r$



degree product, AUC = 0.31

3.4 Other link prediction techniques

There are two other major classes of link prediction algorithms, both of which use *global* information in order to derive a score(i,j), and are often more complicated than most topological predictors.

Model-based predictors are a broad class of algorithms that rely on models of large-scale network structure, e.g., by decomposing the network into modules or communities, to make predictions about missing links. The predictions are produced by one of two strategies: likelihood or optimization.

Likelihood-based predictors are basically structured random-graph models, which estimate a probabilistic model of graphs $\Pr(G \,|\, \theta)$, much like the Erdős-Rényi random graph and the configuration model do, but with more elaborate structure. The advantage of these methods is that by learning the model from data, they also learn a model of the existence of individual edges $\Pr(i \to j \,|\, \theta)$, which we can exploit as a score function for the unconnected pairs. If the model fits the data well, then we can predict that an unconnected pair should be connected if it has a high probability of connection under the model. A very popular approach within this family is the stochastic block

model, or one of its many variants.

Optimization-based predictors are based on algorithms that maximize some measure of the quality of the large-scale decomposition of the network. We can convert this objective function into a score function for unconnected pairs by asking how much adding that edge would improve its quality function.

Embedding-based predictors are another broad class of algorithms, which are based on the idea that close proximity of an unconnected pair within an "embedded space" is indicative of a missing link. To create this embedded space, the algorithms first assign positions to nodes within some $\mathbb{R}^d$ space, for d-dimensions, via some graph embedding algorithm. The way these positions are assigned determines the particular assumptions about structure, and there are many different approaches, e.g., DeepWalk or node2vec; most attempt to preserve "local" structure, meaning that pairs of nodes that are connected in G have small proximities in the embedded space. Given such an "embedding," we can then simply sort the unconnected pairs by some distance measure (there are many) within the space.

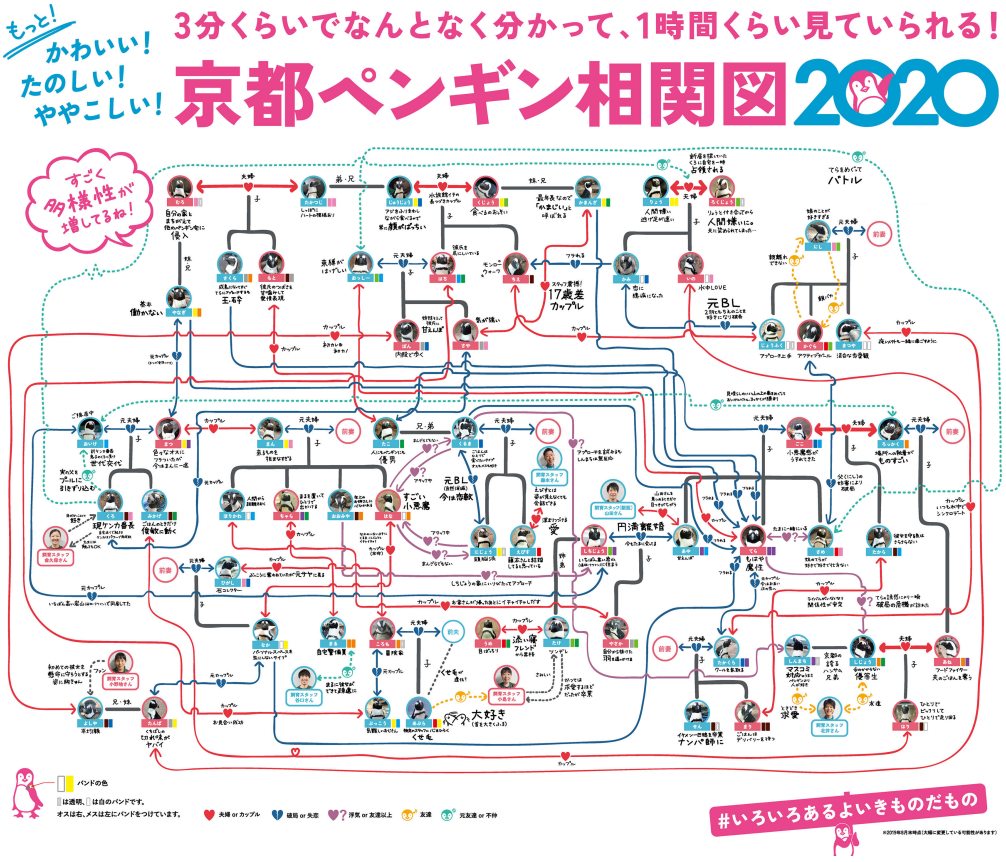
Both of these families can produce good predictions of missing links, but something called the No Free Lunch Theorem[13] basically guarantees that no one algorithm can be best across all inputs, and some recent work suggests that model-based and topological predictors are good general predictors, especially when used in combination. But, performance also varies across the domain of the network: social networks are generally the easiest, with most predictors performing well, while biological networks are among the hardest.[14]

[13]See Wolpert and Macready, "No Free Lunch Theorems for optimization." *IEEE Transactions on Evolutionary Computation* **1**(1), 67–82 (1997)

[14]See Ghasemian et al., "Stacking Models for Nearly Optimal Link Prediction in Complex Networks." *Proc. Natl. Acad. Sci. USA* **117**(38), 23393–23400 (2020).

3.5 Bonus material: The Kyoto Aquarium's Penguins

As a more realistic, but also more complicated example of node attribute prediction, 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.[15] 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 may not equal the neighborhood set $\nu(j)$ for some j, because the edge points in one direction, but

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

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.