

## 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 use one to predict the other.

**Why are networks incomplete?** Often, it is expensive or impractical to collect complete network data, and we typically have to work with what we can get. Most basically, there are  $\Theta(n^2)$  potential pairwise interactions among  $n$  nodes, and identifying with certainty which of those pairs are definitely connected, or definitely not connected, might require checking them all.

In biological networks, collecting data on interactions often requires 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. And, 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 even then,  $\Theta(n^2)$  tends to be a very large number, despite individual experiments being 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. Online 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, won't include interactions that occurred on a different platform.

In networks, there are three main types<sup>1</sup> of missing data prediction tasks:

1. **Predicting missing node attributes:**

Given a network  $G = (V, E)$  and a partial annotation of nodes  $\vec{x}$  by some attribute (categorical or scalar), guess the missing attribute values.

2. **Predicting missing links:**

Given list of nodes and a partial list of edges  $G_o = (V, E_o)$ , guess which unconnected pairs  $i, j$  among  $V \times V - E_o$  are in fact missing connections.

3. **Predicting missing link attributes:**

Given a network  $G = (V, E)$  and a partial annotation of edges  $\vec{w}$  by some attribute (categorical, sign, or scalar), guess the missing attribute values.

In this lecture, we will learn about the general parameters of the first two prediction tasks, describe a few simple approaches for solving them, and learn how to assess their accuracy.

---

<sup>1</sup>A fourth type is to predict missing nodes, but this task is substantially more complicated than predicting individual missing edges. In a way, predicting missing nodes is akin to extrapolation, a fundamentally hard task, while predicting missing links or node attributes is akin to interpolation.

## 2 Predicting missing node attributes

Suppose  $G = (V, E)$  is a fully observed network (no errors or missing links) whose nodes are annotated with some metadata  $\vec{x}$ , such that  $x_i$  gives the value of the  $i$ th node's metadata. These metadata attributes might be *categorical*, representing a node's biological function, e.g., catalysis or transport<sup>2</sup> in the case of proteins, or they might be *scalar*, representing a node's abundance or body mass in the case of species. However, because collecting data is expensive, some nodes are labeled with a missing value  $\emptyset$ , and we would like to replace these with reasonable guesses, using only the network  $G$  and the labels we do have  $\vec{x}$ .

In this setting, we imagine a full labeling  $\vec{x}_*$  that we only observe partially, and a *missingness function*  $f$  that chooses which subset we observe  $\vec{x} = f(\vec{x}_*)$ . Knowing something about the structure of  $f$ , i.e., how it tends to choose which node labels it provides 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.

**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

$$\text{if } x_i = \emptyset, \quad x_i = \text{Uniform}(\vec{x} - \emptyset) \quad , \quad (1)$$

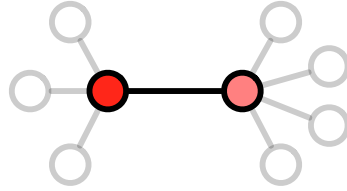
that is, we “impute” each missing label as a uniform draw from the empirical distribution of non-missing labels. If some correlation among  $G$  and  $\vec{x}$  exist, then we can likely improve over this algorithm. There are, of course, many ways we could do this, and these largely fall into two categories: *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  $i$ .

### 2.1 Assortative mixing and local smoothing

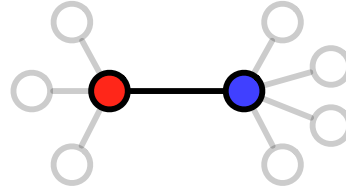
A common feature of many networks is that node annotations are **assortative**,<sup>3</sup> meaning that given an edge  $(i, j)$ , the attributes of those nodes  $x_i$  and  $x_j$  will tend to be more similar to each other than will the attributes of an unconnected pair  $i, j$ . In other words, attributes 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.

<sup>2</sup>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/>.

<sup>3</sup>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

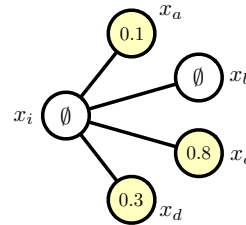
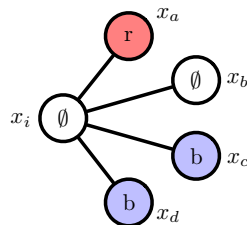
*predict a missing attribute to be the average (scalar) or  
most common (categorical) value of its neighbors' non-missing attributes.*

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  $\vec{x}$  the labels that are local to  $i$ , denoted  $\{x_{\nu(i)}\}$ . 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  $\{x_{\nu(i)}\} - \emptyset$ . Hence, we say

$$\text{if } x_i = \emptyset, \quad x_i = \begin{cases} \text{mean}(\{x_{\nu(i)}\} - \emptyset) & \text{if } x \text{ is scalar} \\ \text{mode}(\{x_{\nu(i)}\} - \emptyset) & \text{if } x \text{ is categorical} \end{cases} .$$

If *all* of  $i$ ’s neighbors have missing values, and thus  $\{x_{\nu(i)}\} = \emptyset$ , we can revert to the baseline prediction algorithm of Eq. (1). And finally, predictions for multiple nodes with missing values are made in parallel, rather than in sequence.

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



## 2.2 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. 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  $x_i \in \{1, 2, \dots, c\}$ , i.e., there are  $c$  distinct values a metadata variable can take. (Imagine simply relabeling the actual values as  $1 \dots c$  for convenience.) Each input we apply our algorithm 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.<sup>4</sup>

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

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

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 detailed information on the behavior of a prediction algorithm. Often, however, we want to summarize the contents of  $\mathcal{C}$  in a single number that describes the algorithm's overall performance. There are *many* different ways to summarize a  $\mathcal{C}$  matrix, and each emphasizes slightly different aspects of an algorithm's behavior.<sup>5</sup> A common measure is the **accuracy**, defined as

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

where  $N$  is the number of inputs on which a prediction was made. But warning: accuracy 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  $\text{ACC} = 0.90$ , but is correct 0% of the time on the minority label.

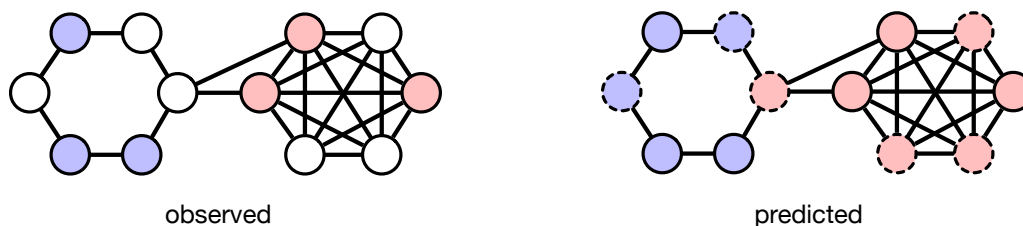
**Scalar values.** When metadata values are scalar, we can simply (i) make a scatter plot of the predicted  $x_i$  vs. the actual  $x_i$  (which is comparable to the confusion matrix), and (ii) report the  $r^2$  correlation (or similar summary statistic) between the two (which has similar weaknesses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

<sup>4</sup>When  $c = 2$  (binary classification), we can use the language of true and false, positives and negatives.

<sup>5</sup>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](https://en.wikipedia.org/wiki/Confusion_matrix)

## 2.3 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.



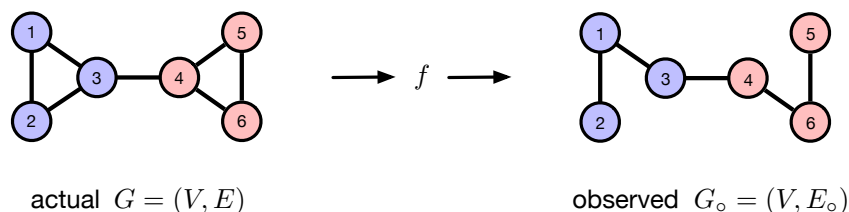
Applying the local smoothing algorithm produces the network on the right (do you see why, for each node?), which makes one mistake. The resulting confusion matrix is then

		actual	
		r	b
pred.	r	3	1
	b	0	2

corresponding to an accuracy of  $\text{ACC} = 5/6 = 0.83$ .

## 3 Predicting missing links

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



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 don't. Instead, our task is to sort among all the 0s of the adjacency matrix (the observed “non-edges”) to find those that should be 1s. Because most real-world networks are sparse, this is like searching for  $O(n)$  needles in a  $\Theta(n^2)$  haystack. Most link prediction methods accomplish this task by defining a *score* function over these many unconnected pairs  $i, j \in X$  such that better-scoring pairs are more likely to be missing links.

**A baseline algorithm.** In the absence of any information about  $f$  or about any patterns within  $E_o$ , a simple **baseline prediction** algorithm is to say

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

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  $\text{score}(i, j)$  values, we obtain a score table:

$i$	$j$	$\text{score}(i, j)$
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  $\text{Uniform}(0, 1)$ , and the ordering is arbitrary among tied scores. 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 likely to be the missing links than any other pair.

If some correlation exists between the observed edges  $E_o$  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 are simple functions of the observed network topology, e.g., counts of edges, measures of overlapping node sets, and measures that summarize a network's structure. Most commonly, these are functions of the joint local network structure of the pair of nodes  $i, j$  being scored. 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.

A common feature of many networks is that edges themselves are **assortative**, meaning that they tend to cluster together. Just as it did for node attributes, this pattern allows us to construct another 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 operationalize such an idea. 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 Section 2.1, let  $\nu(i)$  return the set of neighbors of node  $i$ . The Jaccard coefficient is defined as

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

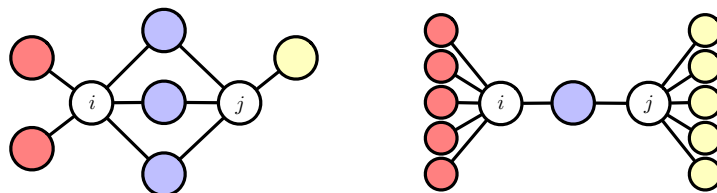
that is, the fraction of the neighbors they have in common out of the set that either of them have. In this way, two nodes with many common neighbors will have a higher Jaccard coefficient, which also counts the number of open triads between  $i, j$  (recall Lecture 2). Adding an edge between  $i, j$  would “close” each of those triangles, thereby increasing their local clustering.

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

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

where the  $\text{Uniform}(0, \epsilon)$  part adds a small amount of noise in order to break ties randomly without changing the relative ordering outside those ties.

Consider the following two little 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, being only 0.091.

A second local topological predictor is the **degree product**, which embodies the idea from random graphs (Lecture 3) that nodes with high degrees are likely themselves to be connected. 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 little examples?

Returning to the first example in this section, from which we removed two 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( $i, j$ )	$i$	$j$	degree product score( $i, j$ )
<b>4</b>	<b>5</b>	$1/2 + r$	1	4	$4 + r$
<b>2</b>	<b>3</b>	$1/2 + r$	1	6	$4 + r$
3	6	$1/3 + r$	3	6	$4 + r$
1	4	$1/3 + r$	1	5	$2 + r$
1	5	$r$	<b>2</b>	<b>3</b>	$2 + r$
1	6	$r$	2	6	$2 + r$
2	6	$r$	2	4	$2 + r$
2	4	$r$	3	5	$2 + r$
2	5	$r$	<b>4</b>	<b>5</b>	$2 + r$
3	5	$r$	2	5	$1 + r$

where the missing links  $Y = \{(2, 3), (4, 5)\}$  are highlighted, and links with the same score are ordered arbitrarily. The Jaccard link predictor does very well at assigning these missing links high scores compared to the non-missing links, while the degree product seems very poor. To understand how these compare to the baseline, or 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,<sup>6</sup> a context-agnostic measure of its ability to distinguish a missing link  $i, j \in Y$  (a true positive) from a non-edge  $X - Y$  (a true negative). The AUC has two other attractive properties: (1) it's *scale invariant*, meaning it's not dependent on the scores themselves, and instead only depends on their relative values, and (2) it's *threshold invariant*, meaning it's 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

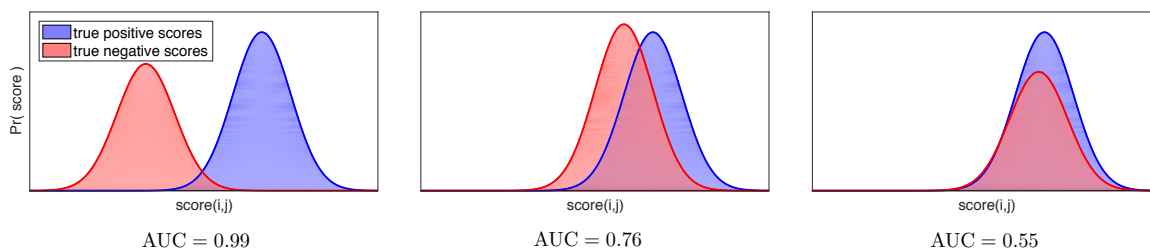
$$\text{AUC} = \Pr[\text{score}(\text{true positive}) > \text{score}(\text{true negative})] \quad . \quad (7)$$

That is, if we choose a uniformly random missing edge  $a, b \in Y$  and a uniformly random non-edge  $c, d \in X - Y$ , the AUC is the probability that  $\text{score}(a, b)$  is higher than  $\text{score}(c, d)$ . The figure

<sup>6</sup>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. See [https://en.wikipedia.org/wiki/Receiver\\_operating\\_characteristic](https://en.wikipedia.org/wiki/Receiver_operating_characteristic)



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.



If the algorithm is no better than guessing at random, then  $AUC = 0.5$  and it's equally likely that a true negative has the higher score than a true positive, and vice versa. The baseline predictor of Eq. (4), which assigns a uniformly random value to every candidate, has exactly this behavior and thus has an accuracy of  $AUC = 0.5$ . Hence, any algorithm with  $AUC > 0.5$  does better than chance. 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).

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.<sup>7</sup> 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  $\ell$ th row, and then sliding it up, or down the table. This line represents the prediction threshold: all the rows  $\ell$  and above we predict to be missing links, while rows  $\ell + 1$  and below it are predicted 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  $\ell$ ), and the FPR is the fraction of all the actual non-missing links (non-edges) that are in the same set. Plotting  $TPR(\ell)$  vs.  $FPR(\ell)$  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  $\tau$  that gives the *actual* status of the  $\ell$ th row (pair  $i,j$ ) as either a true positive ( $\tau_\ell = 1$ ) or true negative ( $\tau_\ell = 0$ ), and then two columns, one for  $TPR(\ell)$  and one for  $FPR(\ell)$ . If a predictor performs well, then in the  $\tau$  column, the 1s will tend to be located higher up in the table, while if

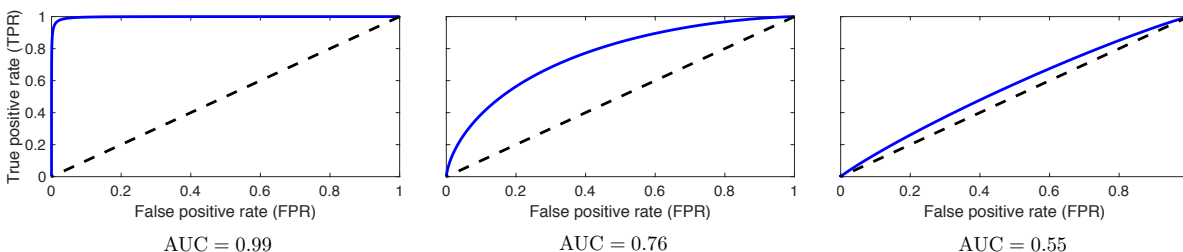
<sup>7</sup>A less common but more scalable method applies Eq. (7) directly, by sampling many pairs of true positives and true negatives to estimate the fraction of draws for which the former is assigned a higher score than the latter.

it is no better than baseline, the 1s will be scattered uniformly at random within this column.<sup>8</sup>

Once the  $\tau$  columns 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  $\tau$ ), 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  $\tau$ ). The  $\text{TPR}(\ell)$  and  $\text{FPR}(\ell)$  are then defined as

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

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  $\text{TPR} = \text{FPR}$  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 box-rule approximation<sup>9</sup> technique:

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

where we define  $\text{FPR}(0) = 0$ .

### 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

<sup>8</sup>If the 1s tend to be stacked at the bottom of the  $\tau$  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).

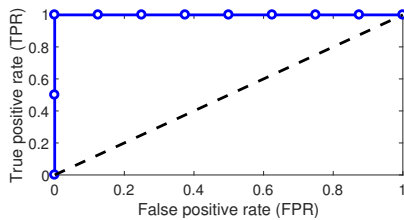
<sup>9</sup>Recall the box rule for numerical integration: given a function  $f(x)$ , a range  $[x_s, x_t]$ , and an increment  $\Delta x$ , the integral  $\int_{x_s}^{x_t} f(x) dx \approx \sum_{x=x_s}^{x_t} f(x) \times \Delta x$ . The trapezoid rule is slightly more accurate, although the difference is negligible when  $\Delta x$  is small, as is the case in link prediction (do you see why?).

status  $\tau$ , 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.)

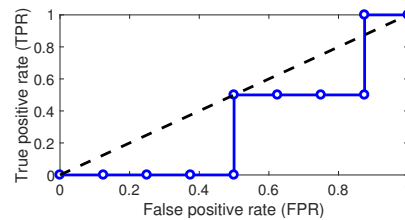
From these, we can plot the corresponding ROC curves, and apply Eq. (9) to calculate the AUC for each predictor. Unsurprisingly, the Jaccard predictor achieves perfect performance, with an  $AUC = 1.00$ . In this network, the degree-product predictor is in fact *worse* than guessing at random, producing 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  $AUC = 0.37$ .

$i$	$j$	$\tau_k$	$\text{TPR}_\ell$	$\text{FPR}_\ell$	Jaccard $\text{score}(i, j)$
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	$r$

$i$	$j$	$\tau_k$	$\text{TPR}_\ell$	$\text{FPR}_\ell$	degree product $\text{score}(i, j)$
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	$1 + r$



Jaccard coefficient,  $AUC = 1.00$



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  $\text{score}(i, j)$ , 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  $\Pr(i \rightarrow j | \theta)$ , 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; 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<sup>10</sup> 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.<sup>11</sup>

---

<sup>10</sup>See Wolpert and Macready, “No Free Lunch Theorems for optimization.” *IEEE Transactions on Evolutionary Computation* 1(1), 67–82 (1997)

<sup>11</sup>See Ghasemian et al., “Stacking Models for Nearly Optimal Link Prediction in Complex Networks.” Preprint, <https://arxiv.org/abs/1909.07578> (2019).