1 Predicting Missing Links in Networks

Recall that 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 and we only get to observe them for part of their lifetime. When edges are the things that are missing, link prediction aims to identify those node pairs that are most likely to be missing links (or, future connections), based on their correlation with the links we do observe.

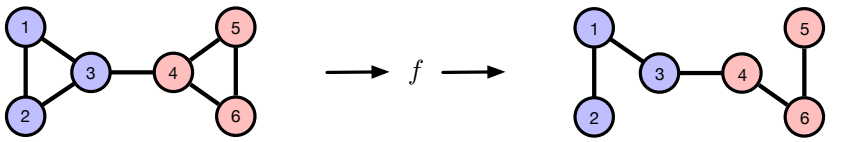
Link prediction serves three primary functions in network data science:

- *Filling in missing or future connections*, because the observed network G° is known to represent an incomplete set of interactions.
- *Marshaling scarce resources* for edge discovery, in the setting where each pair of nodes i, j has to be measured directly, e.g., by a biological experiment or a social observation.
- *Comparing models*, by asking which model is better at "out of sample" predictions of edges. In this way, link prediction is akin to cross validation in machine learning, in which we divide the edge set E into a "training" set (observed links) and a "test" set (missing links), and then compare models in how well they can learn to predict the missing links based only on the observed links.[1]

In this setting, we assume that we have a complete account of a set of nodes V, but an incomplete account of the edges among them. Letting E denote the set of actual edges, and $E^\circ \subset E$ be the subset of them that we observe, we can define the observed network as $G^\circ = (V, E^\circ)$ and the actual network as $G = (V, E)$. Just as with missing node attributes, these is some *missingness function* f that chooses which subset of edges E° we actually observe. 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.

[1] Although this very analogy between link prediction and cross validation is commonly invoked, it is not mathematically exact. In the traditional setting where cross validation is well-understood mathematically, we assume we have iid draws of vectors from some underlying high-dimensional data generating process. Choosing a random partition of this set into two subgroups is mathematically well-defined because of the iid assumption. In networks, however, we lose the independence property, and a random partition of the edges may no longer be mathematically meaningful, e.g., when edges only ever appear in bundles or when edges perform some system-level function. Consider the example of whether a random partition of your genome's gene interactions would still be a functioning genome. Understanding how to do cross validation properly in a network setting is an active area of research.



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

Mathematically, predicting missing links is a much harder problem than predicting missing attributes of either nodes or edges. With missing 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 $i, j \in Y$, 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. The idea of guessing at random provides a simple baseline algorithm—if we can do better than this baseline, then we're getting somewhere—and this algorithm is equally likely to assign any particular score to each of the unconnected pairs.

The baseline predictor. In the absence of any information about f or about any patterns within E° and their relationship to the missing links in X, the simplest **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. We can formalize this notion mathematically, as follows:

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

That is, the baseline assigns 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 $X = \{(2,3),(4,5)\}$ 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 X, then we can likely improve considerably over this baseline. Just like with predicting missing node attributes, there are many ways we could do this. In practice, there are three approaches:

- **Topological predictors** are simple functions of the joint local network structure of the pair of nodes i, j being scored, e.g., their degrees, measures of overlapping neighborhoods, geodesic distance, and other measures that summarize a network's structure.
- **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.
- **Embedding-based predictors** are second broad class of algorithms that assign nodes to locations in a d-dimensional latent space in such a way that nodes are "close" in the embedded space if they are, or are likely to be, connected.

In this lecture, we'll focus primarily on topological predictors, and then briefly consider the remaining two in Section 2 before considering advanced techniques.

1.1 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.[2]

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 operationalize such an idea into a specific 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

$$\text{Jaccard}(i,j) = \frac{|\nu(i) \cap \nu(j)|}{|\nu(i) \cup \nu(j)|} \ , \tag{2}$$

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.

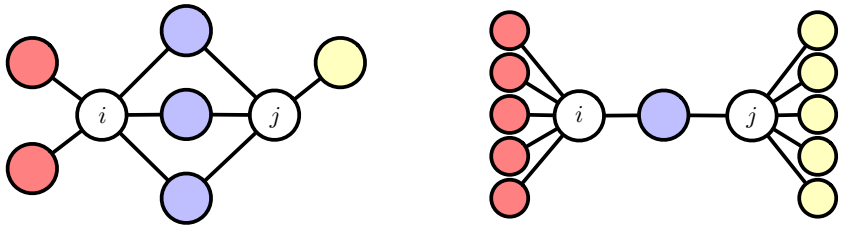
To turn this pairwise network measure into a prediction algorithm, we say

$$\text{score}(i,j) = \text{Jaccard}(i,j) + \text{Uniform}(0,\epsilon) \ , \tag{3}$$

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.

[2]There are an enormous number of topological predictors for missing links. Examples include the number of common neighbors, shortest-path betweenness centrality, the Leicht–Holme–Newman index, personalized page rank, shortest path length, the mean neighbor entries within a low rank approximation, the Jaccard coefficient, the Adamic–Adar index, the resource allocation index, the dot product of columns i and j in a low rank approximation via a singular value decomposition, the local clustering coefficients of i and j, the average neighbor degrees, and any number of "centrality" measures.



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?

Let us return to the example network on the first page of this lecture, from which we removed two edges, and then apply the Jaccard and degree-product predictors to the observed network G'. Doing so produces 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 $X = \{(2,3),(4,5)\}$ 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.

1.2 Measuring performance: the AUC

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

To summarize the performance of a link prediction algorithm, we can use the **AUC** statistic,[3] [4] a context-agnostic measure of its ability to distinguish a missing link $i, j \in X$ (a true positive, or TP) from a non-edge $Y - X$ (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

$$\text{AUC} = \Pr[\ \text{score(TP)} > \text{score(TN)}\ ] \quad . \tag{4}$$

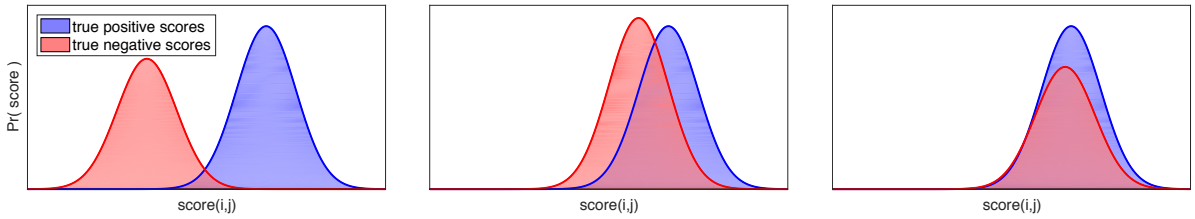
That is, if we choose a uniformly random missing edge $a, b \in X$ (TP) and a uniformly random non-edge $c, d \in Y - X$ (TN), the AUC is the probability that score(a, b) is higher than score(c, d).[5] The figure below illustrates this idea, for three hypothetical predictors. On the left, the predictor 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.

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. (1), 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

[3] 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.

[4] There are other ways to quantify accuracy in this setting, each of which has a slightly different way of weighting the cost of different types of errors. Common choices include the F1-measure, which is the harmonic mean of the precision and recall, or the entire precision-recall curve. These alternatives are less context agnostic than the AUC, and hence may provide a better guide to performance in particular settings.

[5] 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.



AUC = 0.99 AUC = 0.76 AUC = 0.55

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.[6] 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.[7]

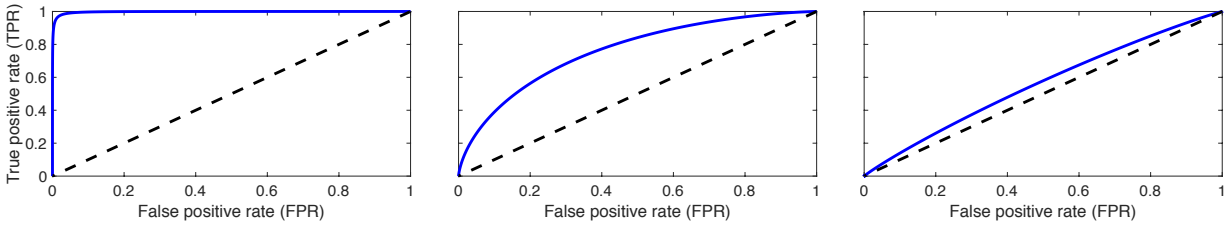
Once the τ column has been added, we can fill in the TPR and FPR columns simultaneously in a single scan down the table. Let $\mathcal{T} = |X|$ be the total number of true positives (missing links; equal to the column sum of τ), and let $\mathcal{F} = |Y - X|$ be the total number of true negatives (non-edges;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

$$\text{TPR}(\ell) = \frac{1}{\mathcal{T}} \sum_{k=1}^{\ell} \tau_k \qquad \text{FPR}(\ell) = \frac{1}{\mathcal{F}} \sum_{k=1}^{\ell} 1 - \tau_k \ . \tag{5}$$

[6] We can also estimate the AUC via Monte Carlo, by evaluating Eq. (4) 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.

[7] 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).

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.



AUC = 0.99 AUC = 0.76 AUC = 0.55

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[8] technique, running down the elements of the TPR and FPR vectors:

$$\text{AUC} = \sum_{\ell=1}^{|Y|} \text{TPR}(\ell) \times [\text{FPR}(\ell) - \text{FPR}(\ell-1)] \quad , \tag{6}$$

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.

1.3 An example

To see how this works in practice, let's consider again the Jaccard and degree-product link predictors from Section 1.1. 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.)

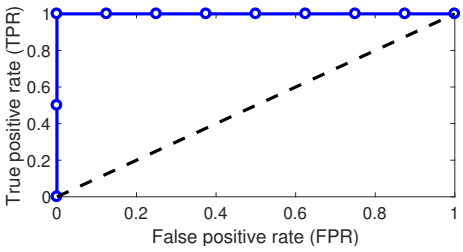
From these, we can plot the corresponding ROC curves, and apply Eq. (6) to calculate the AUC for each predictor. Unsurprisingly, the Jaccard predictor achieves perfect performance, with an AUC = 1.00. In this network, the degree-product predictor is in fact *worse* than guessing at random, producing 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

[8] 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?).

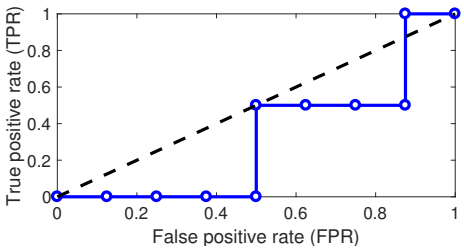
these lead to slightly different TPR and FPR functions. The right thing to do here is to compute an *average AUC*, when these ties are broken randomly, which produces a more reliable estimate of the AUC. The highest value possible, if the ties break luckily for the missing links is 0.62, while the lowest, 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

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	$1 + r$



Jaccard coefficient, AUC = 1.00



degree product, AUC = 0.31

2 Advanced approaches

Warning: this material is rough and still under development.

Just as with node attribute prediction, using the network to predict missing links only works if there are correlations between the missing edges and the observed edges. And, although we can make the predictions work based on those correlations alone, we will almost always get better predictions if we have some understanding of both why edges are missing (the missingness function f) and why two nodes should be connected in the first place (the data generating process).

Modern approaches to link prediction typically recast the problem as a machine learning task in one of two ways:

- Specify a set of features for each *pair* of nodes i, j—the pair's features may include concatenating the two nodes' attributes $\mathbf{x}_i \, \mathbf{x}_j$, concatenating their various node-level summary statistics, e.g., $k_i k_j$, and deriving features from the pair itself, e.g., their geodesic distance ℓ_{ij} in the graph excluding any connection (i, j)—and then train a binary classifier on the correlation between the pair-features and the existence or not of an edge.

  If the underlying data generating process for edges is fairly homogeneous across the network itself, then this approach can work very well; however, if the data generating mechanism varies across the network or the network exhibits large-scale structure like communities or groups, then this approach may struggle to capture that variability.

- Learn a probability distribution centered on the observed graph $\Pr(G^\circ \,|\, \hat{\theta})$ and then score unconnected nodes in G° according to their probability of connection under that model $\Pr(i \to j \,|\, \hat{\theta})$.

  These models can capture the kind of heterogeneity that the feature-based approaches miss, but they are often harder to work with because they require specifying the model's structure precisely enough to be estimated from data.

Both of these are *global* prediction algorithms, because they leverage information across the entire network to make each particular prediction, and hence these approaches tend to be more computationally expensive than the simple topological predictors we learned about above. Within these broad categories, two are particularly popular.

Embedding-based predictors are a broad class of algorithms, based on the idea that close proximity of an unconnected pair within an "embedded space" is indicative of a missing link. That is, these algorithms assume node homophily in an inferable latent space explains edge existence, and we can approximate that space by learning an embedding of the observed graph G° into some $\mathbb{R}^d$ space, with d-dimensions.

To create this embedded space, an algorithm will first assign the n nodes of G° to d-dimensional locations within $\mathbb{R}^d$, 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 sort the unconnected pairs by some distance measure (there are many) within the space to make predictions.

Model-based predictors are a second broad class of algorithms, which typically rely on a probabilistic generative model of network structure, e.g., representing the whole graph as being composed of connections between or within a set of modules or communities. That is, we assume the observed data is an instance of a, possibly elaborate, random graph model. The configuration model

is a relatively simple random graph model compared to the typical model used in this approach. Typically, these models assume that edges are independent, conditioned on some latent parameters that govern the particular probability that i, j are connected.

To learn these probabilities, we estimate the graph model's latent parameters from the observed graph G°. The way these parameters interact to determine the conditional probabilities of edges embodies the particular assumptions about structure, and why different pairs of nodes are more or less likely to be connected. By learning the model from the observed data, we also learn a model of the existence of individual edges $\Pr(i \to j \,|\, \hat{\theta})$, which we 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. The most popular approaches within this family are different all variations of the stochastic block model.

Caveats. Both of these families can produce good predictions of missing links, but something called the No Free Lunch Theorem[9] 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.[10]

2.1 Meta-learning our way to missing link predictions

Model stacking (also called stacked generalization) is an ensemble technique that learns to combine weakly predictive base models to construct an optimal predictive distribution.

Stacking begins by training a set of N base learners or level-0 models $\{f_1, f_2, \ldots, f_N\}$ on a common data set X, and then "stacking" a meta-learner or level-1 model on top to learn to synthesize their outputs into a final prediction Y. Level-0 models can be any supervised or unsupervised technique. Each base learner f_i maps an input $x \in X$ to a predicted output $f_i(x) \in \mathbb{R}$, and subsequently, the predictions $f_i(x)$ obtained on the training set are treated as additional training data for the meta-learner model F. Hence, the level-1 model learns to predict the true target $y \in \mathbb{R}$ by combining various level-0 model predictions: $y_{\text{final}} = F(f_1(x), f_2(x), \ldots, f_N(x))$. For scientific applications where interpretability is important, users can select a level-1 model with good interpretability.

There are many choices for the level-1 model: nearly any supervised technique can work. Random forests are an ensemble method that constructs a set of decision trees that convert a feature space

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

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

into an output prediction score. Random forests create an ensemble of trees by bootstrapping the data for each tree and selecting a random subset of features to learn from. They are the most popular choice for the level-1 model in this approach, being both very accurate and highly scalable (cheap to train on commodity hardware).

XGBoost is a similar method to random forests that sequentially builds a set of decision trees, with each learning to correct errors made by the previous tree, via gradient boosting. This model is also accurate and highly scalable. Both of these level-1 choices require some hyperparameter tuning in order to optimize their results, e.g., number of trees, tree depth, and number of features for random forests, and learning rate, tree depth, and number of trees for XGBoost.

What should we use for the level-0 models? In practice, we can use any predictors we like. For networks, it's common to use unsupervised and computationally lightweight topological functions of the input network. At a high level, these functions come in three flavors: global, node-level, and pair-level.

Global predictors are network-level statistics like the number of nodes n, number of edges m, etc., which help contextualize other network features. Node-level predictors are properties of individual nodes, such as their degree k_i, or any centrality measure, local clustering coefficient C_i, etc. For a given pair $i, j \in X$ that we are scoring, we include the node-level predictor for both i and j, as a pair of features. Finally, pairwise predictors are simple or complicated functions of the network relationship of the pair i, j, and can include measures like the number of common neighbors $\mathrm{CN}(i, j)$, etc.

[[This section still being developed]]