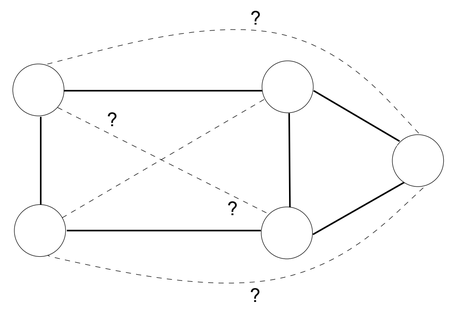
Link prediction via matrix factorization

Aditya Krishna Menon, Charles Elkan; University of California San Diego
{akmenon, elkan}@ucsd.edu

The link prediction problem

Given a graph *G* that is partially observed -- meaning each pair of nodes is either known to have an edge, known to have no edge, or not *known* to have an edge or not -- we wish to predict whether or not edges exist for the unknown status node pairs. Examples include predicting whether a pair of proteins interact, whether a pair of authors cite each others' papers, and whether a pair of countries will go to war with each other.



Challenges for a link prediction method

There are three basic challenges for a link prediction method:

* Linking behaviour can be influenced by both the topological structure of the observed graph (embedded in the adjacency matrix), as well as measured characteristics of the nodes (presented in the form of explicit features). A model should use *both* types of information to maximize predictive accuracy.

* In many applications of link prediction, there are far fewer links than non-links i.e. a **class imbalance** problem. For example, on a dataset of the US electric power grid, there is 1 link formed for every 2000 non-links.

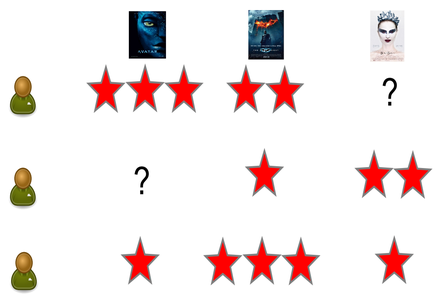
* Real-world networks have the order of millions of nodes, so models should be highly **scalable**.

Classical models for link prediction are affinity scores such as Common-Neighbours, Preferential-Attachment, the Katz measure, *et cetera*. These methods are *unsupervised*, in that they are based on fixed functions of the graph's topological information. *Supervised* methods have also been studied, with popular choices being learning a classifier that combines multiple affinity scores, and/or explicit features for nodes.

The *de-facto* mechanism to handle class imbalance is to undersample the data by throwing away some of the node pairs that do not link, which necessarily loses information.

A latent feature approach

Link prediction can be cast as a **dyadic prediction** problem, as we predict some label (whether or not there is an edge) for a *dyad* or pair of objects (nodes in the graph). A similar problem is **collaborative filtering**, where a popular solution is to learn latent features from the data. We may use a similar approach for link prediction: mathematically, we optimize some loss ℓ on the observed links $\mathcal{O}$ via



$$\min_{U,\Lambda,b} \frac{1}{|\mathcal{O}|} \sum_{(i,j)\in\mathcal{O}} \ell(G_{ij}, L(u_i^T \Lambda u_j + b_i + b_j)) + \Omega(U, \Lambda)$$

where *L* is a link function, such as $L(z) = (1 + e^{-z})^{-1}$, u_i is a **latent feature** for the *i*th node, and b_i is a bias term for the *i*th node. The Ω term is a regularizer to prevent overfitting.

This model can be trained using **stochastic gradient descent (SGD)**, and so is highly scalable. It is also very expressive, capturing latent class methods as a special case, and empirically outperforming affinity scores like Common-Neighbours by virtue of being supervised. However, it is limited in two ways: (i) it only exploits topological information, and (ii) it is not immune to the class imbalance issue. We see now how to fix these problems.

Adding node and edge features

Given node features $x_i \in \mathbb{R}^d$ and edge features $z_{ij} \in \mathbb{R}^{d'}$, we may model:

$$\hat{G}_{ij} = L(u_i^T \Lambda u_j + b_i + b_j + w_1^T x_i + w_2^T x_j + v^T z_{ij})$$

This models the affinity for a pair of nodes as a linear combination of their affinity in latent and explicit feature space. We can equivalently think of the latent features as providing a residual fit on the scores for a supervised learning technique such as logistic regression on the explicit features, corresponding to a sigmoidal link function.

The above suffers from the *propensity problem*: ignoring the influence of latent and edge-features, we would get the same ranking over all nodes. A better alternative is to allow for interactions bewteen the node features, in a model known as *bilinear regression* [Gabriel, Biometrika '98]:

$$\hat{G}_{ij} = L(u_i^T \Lambda u_j + b_i + b_j + x_i^T W x_j + v^T z_{ij})$$

We can now learn latent features in addition to the weights on the explicit features. In practice, it may be necessary to factorize $W = A^T B$ also, since otherwise we may need to learn too many parameters.

Overcoming imbalance by AUC maximization

As mentioned earlier, a key challenge in link prediction is the **class imbalance** problem: the majority of node-pairs do not link with each other. As a result, area under the ROC curve (*AUC*) is a standard performance measure, as it is not influenced by relative size of classes. It measures the number of concordant pairs of +'ve and -'ve examples under a scoring rule *f*:

$$\mathcal{A} \propto \sum_{i\in+, j\in-} \mathbf{1}[f_i - f_j > 0]$$

Classifiers such as logistic regression and SVMs optimize convex approximations to the 0-1 error. The inadequacy of this metric for imbalanced problems explains their susceptibility to class imbalance. However, note that we can equally optimize a convex approximation to the AUC:

$$\hat{\mathcal{A}} \propto \sum_{i\in+, j\in-} \ell(1, f_i - f_j)$$

This can be optimized by SGD: for each update, we just pick a random pair of positive and negative examples. By optimizing the AUC directly, we can overcome the class imbalance issue.

This idea applies equally well when learning latent features (further illustrating the flexibility of this approach). Instead of optimizing square- or log-loss, we can learn latent features that maximize the AUC:

$$\min_{\alpha,\Lambda,W,v} \sum_{(i,j,k)\in\mathcal{D}} \ell(\hat{G}_{ij} - \hat{G}_{ik}, 1) + \Omega(\alpha, \Lambda, W, v)$$

In practice, only a fraction of a single SGD epoch is needed for convergence.

Experimental setup

We ran experiments on six link prediction datasets, from a range of domains: computational biology (Prot-Prot and Metabolic), coauthorship (NIPS and Condmat), political science (Conflict) and general engineering (PowerGrid).

Dataset	Nodes	$\lvert\mathcal{O}^+\rvert$	$\lvert\mathcal{O}^-\rvert$	+ve:−ve	Avg degree
Prot-Prot	2617	23710	6,824,979	1 : 300	9.1
Metabolic	668	5564	440,660	1 : 80	8.3
NIPS	2865	9466	8,198,759	1 : 866	3.3
Condmat	14230	2392	429,232	1 : 179	0.17
Conflict	130	320	16580	1 : 52	2.5
PowerGrid	4941	13188	24,400,293	1 : 1850	2.7

All datasets except Condmat and PowerGrid possess some form of side-information. We report AUC as our performance metric.

Experimental results

We first compare supervised latent features to unsupervised affinity scores. We use the Adamic-Adar, Preferential-Attachment, and the Katz measure. Our latent feature approach outperforms these methods (Fig 1), and especially offers improvement when only a few links are observed in the training data (Fig 2).

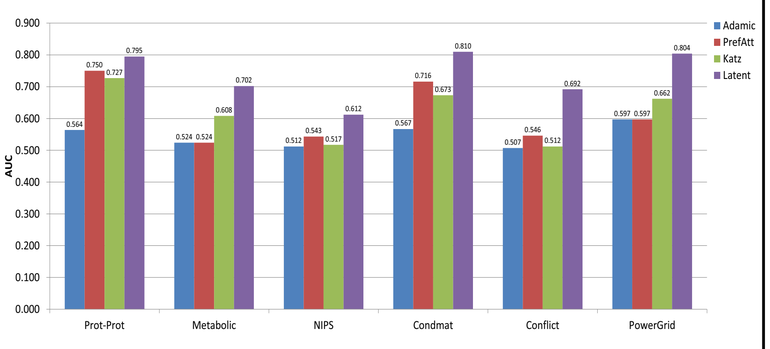


Figure 1: Latent features outperform popular unsupervised scores.

We next evaluate the relative benefits of latent and explicit features. We use unilinear and bilinear regression on the explicit features, and the exact link propagation method (ELP) of [Kashima *et al*, SDM '09]. On several datasets, explicit features are more predictive than latent features alone. The combined latent+explicit feature model achieves the best of both worlds. We note also that bilinear regression offers significant improvement over the unilinear variant.

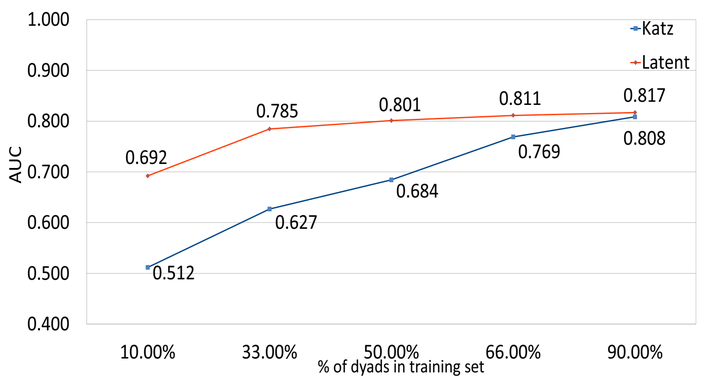


Figure 2: Unsupervised scores underperform with fewer training dyads.

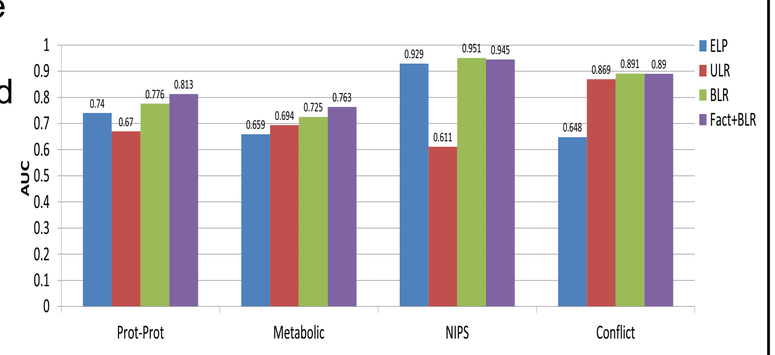


Figure 3: Combining latent and explicit features can improve performance.

We finally compare using a standard regression loss to the ranking loss when learning latent features. We see that the ranking loss performs at least as good as the regression loss, offering improvements when the dataset is especially imbalanced, such as on the PowerGrid dataset.

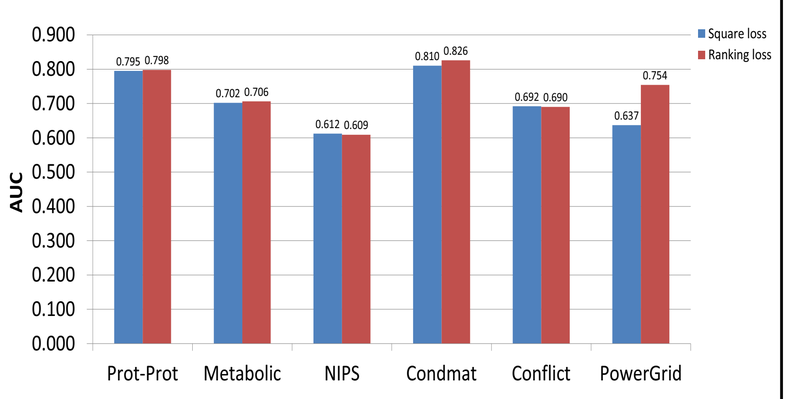


Figure 4: Ranking loss can significantly improve performance when data is very imbalanced.